

AN INTRODUCTION TO CP/M FEATURES AND FACILITIES

Copyright (c) 1979 by Exidy Inc. All Rights reserved. No part of this publication may be reproduced, transmitted, transcribed, stored in a retrieval system, or translated into any language or computer language, in any form or by any means, electronics, mechanical, magnetic, optical, chemical, manual or otherwise, without the prior written permission of Exidy Inc., 390 Java Drive, Sunnyvale, California, 94086.

Disclaimer

Exidy Inc. makes no representations or warranties with respect to the contents hereof and specifically disclaims any implied warranties of merchantability or fitness for any particular purpose. Further, Exidy Inc. reserves the right to revise this publication and to make changes from time to time in the content hereof without obligation of Exidy Inc. to notify any person of such revision or changes.



23 SEP. 1981

RC-103

Table of Contents

Section	Page
1. INTRODUCTION	1
2. FUNCTIONAL DESCRIPTION OF CP/M	3
2.1. General Command Structure	3
2.2. File References	3
3. SWITCHING DISKS	6
4. THE FORM OF BUILT-IN COMMANDS	7
4.1. ERA afn cr	7
4.2. DIR afn cr	8
4.3. REN ufn1=ufn2 cr	8
4.4. SAVE n ufn cr	9
4.5. TYPE ufn cr	9
5. LINE EDITING AND OUTPUT CONTROL.....	11
6. TRANSIENT COMMANDS	12
6.1. STAT cr	13
6.2. ASM ufn cr	16
6.3. LOAD ufn cr	17
6.4. PIP cr	18
6.5. ED ufn cr	25
6.6. SYSGEN cr	27
6.7. SUBMIT ufn parm#1 ... parm#n cr	28
6.8. DUMP ufn cr	30
6.9. MOVCPM cr	30
7. BDOS ERROR MESSAGES	33
8. OPERATION OF CP/M ON THE MDS	34

AND INFORMATION

SEP 23 1951

LABORATORY



[Faint, mostly illegible text covering the majority of the page, likely bleed-through from the reverse side.]

1. INTRODUCTION

CP/M is a monitor control program for microcomputer system development which uses IBM-compatible flexible disks for backup storage. Using a computer mainframe based upon Intel's 8080 microcomputer, CP/M provides a general environment for program construction, storage, and editing, along with assembly and program check-out facilities. An important feature of CP/M is that it can be easily altered to execute with any computer configuration which uses an Intel 8080 (or (Zilog Z-80) Central Processing Unit, and has at least 16K bytes of main memory with up to four IBM-compatible diskette drives. A detailed discussion of the modifications required for any particular hardware environment is given in the Exidy Inc. document entitled "CP/M System Alteration Guide". Although the standard Exidy version operates on a single-density Intel MDS 800, several different hardware manufacturers support their own input-output drivers for CP/M. -

The CP/M monitor provides rapid access to programs through a comprehensive file management package. The file subsystem supports a named file structure, allowing dynamic allocation of file space as well as sequential and random file access. Using this file system, a large number of distinct programs can be stored in both source and machine executable form.

CP/M also supports a powerful context editor.

CP/M is logically divided into several distinct parts:

BIOS	Basic I/O System (hardware dependent)
BDOS	Basic Disk Operating System
CCP	Console Command Processor
TPA	Transient Program Area

The BIOS provides the primitive operations necessary to access the diskette drives and to interface standard peripherals (TTY, CRT, paper tape reader/punch, and user-defined peripherals), and can be tailored by the user to any particular hardware environment by "patching" this portion of CP/M. The BDOS provides disk management by controlling one or more disk drives containing independent file directories. The BDOS implements disk allocation strategies which provide fully dynamic file construction while minimizing head movement across the disk during access. Any particular file may contain any number of records, not exceeding the size of any single disk. In a standard CP/M system, each disk can contain up to 64 distinct files. The

BDOS has entry points which include the following primitive operations which can be programmatically accessed:

SEARCH	Look for a particular disk file by name.
OPEN	Open a file for further operations.
CLOSE	Close a file after processing.
RENAME	Change the name of a particular file.
READ	Read a record from a particular file.
WRITE	Write a record onto the disk.
SELECT	Select a particular disk drive for further operations.

*console
command
processor* →

The CCP provides symbolic interface between the user's console and the remainder of the CP/M system. The CCP reads the console device and processes commands which include listing the file directory, printing the contents of files, and controlling the operation of transient programs, such as assemblers, editors, and debuggers. The standard commands which are available in the CCP are listed in a following section.

The last segment of CP/M is the area called the Transient Program Area (TPA). The TPA holds programs which are loaded from the disk under command of the CCP. During program editing, for example, the TPA holds the CP/M text editor machine code and data areas. Similarly, programs created under CP/M can be checked out by loading and executing these programs in the TPA.

It should be mentioned that any or all of the CP/M component subsystems can be "overlayed" by an executing program. That is, once a user's program is loaded into the TPA, the CCP, BDOS, and BIOS areas can be used as the program's data area. A "bootstrap" loader is programmatically accessible whenever the BIOS portion is not overlayed; thus, the user program need only branch to the bootstrap loader at the end of execution, and the complete CP/M monitor is reloaded from disk.

It should be reiterated that the CP/M operating system is partitioned into distinct modules, including the BIOS portion which defines the hardware environment in which CP/M is executing. Thus, the standard system can be easily modified to any non-standard environment by changing the peripheral drivers to handle the custom system.

2. FUNCTIONAL DESCRIPTION OF CP/M.

The user interacts with CP/M primarily through the CCP, which reads and interprets commands entered through the console. In general, the CCP addresses one of several disks which are online (the standard system addresses up to four different disk drives). These disk drives are labelled A, B, C, and D. A disk is "logged in" if the CCP is currently addressing the disk. In order to clearly indicate which disk is the currently logged disk, the CCP always prompts the operator with the disk name followed by the symbol ">" indicating that the CCP is ready for another command. Upon initial start up, the CP/M system is brought in from disk A, and the CCP displays the message

xxK CP/M VER m.m

where xx is the memory size (in kilobytes) which this CP/M system manages, and m.m is the CP/M version number. All CP/M systems are initially set to operate in a 16K memory space, but can be easily reconfigured to fit any memory size on the host system (see the MOVCPM transient command). Following system signon, CP/M automatically logs in disk A, prompts the user with the symbol "A>" (indicating that CP/M is currently addressing disk "A"), and waits for a command. The commands are implemented at two levels: built-in commands and transient commands.

2.1. GENERAL COMMAND STRUCTURE.

Built-in commands are a part of the CCP program itself, while transient commands are loaded into the TPA from disk and executed. The built-in commands are

ERA	Erase specified files.
DIR	List file names in the directory.
REN	Rename the specified file.
SAVE	Save memory contents in a file.
TYPE	Type the contents of a file on the logged disk.

Nearly all of the commands reference a particular file or group of files. The form of a file reference is specified below.

2.2. FILE REFERENCES.

A file reference identifies a particular file or group of files on a particular disk attached to CP/M. These file references can be either "unambiguous" (ufn) or "ambiguous" (afn). An unambiguous file reference uniquely identifies a single file, while an ambiguous file reference may be

satisfied by a number of different files.

File references consist of two parts: the primary name and the secondary name. Although the secondary name is optional, it usually is generic; that is, the secondary name "ASM," for example, is used to denote that the file is an assembly language source file, while the primary name distinguishes each particular source file. The two names are separated by a "." as shown below:

pppppppp.sss

where pppppppp represents the primary name of eight characters or less, and sss is the secondary name of no more than three characters. As mentioned above, the name

pppppppp

is also allowed and is equivalent to a secondary name consisting of three blanks. The characters used in specifying an unambiguous file reference cannot contain any of the special characters

< > . , ; : = ? * []

while all alphanumerics and remaining special characters are allowed.

An ambiguous file reference is used for directory search and pattern matching. The form of an ambiguous file reference is similar to an unambiguous reference, except the symbol "?" may be interspersed throughout the primary and secondary names. In various commands throughout CP/M, the "?" symbol matches any character of a file name in the "?" position. Thus, the ambiguous reference

X?Z.C?M

is satisfied by the unambiguous file names

XYZ.COM

and

X3Z.CAM

Note that the ambiguous reference

.

is equivalent to the ambiguous file reference

?????????.???

while

and pppppppp.*

*.sss

are abbreviations for

pppppppp.???

and

?????????.sss

respectively. As an example,

DIR *.*

is interpreted by the CCP as a command to list the names of all disk files in the directory, while

DIR X.Y

searches only for a file by the name X.Y Similarly, the command

DIR X?Y.C?M

causes a search for all (unambiguous) file names on the disk which satisfy this ambiguous reference.

The following file names are valid unambiguous file references:

X	XYZ	GAMMA
X.Y	XYZ.COM	GAMMA.1

As an added convenience, the programmer can generally specify the disk drive name along with the file name. In this case, the drive name is given as a letter A through Z followed by a colon (:). The specified drive is then "logged in" before the file operation occurs. Thus, the following are valid file names with disk name prefixes:

A:X.Y	B:XYZ	C:GAMMA
Z:XYZ.COM	B:X.A?M	C:*.ASM

It should also be noted that all alphabetic lower case letters in file and drive names are always translated to upper case when they are processed by the CCP.

3. SWITCHING DISKS.

The operator can switch the currently logged disk by typing the disk drive name (A, B, C, or D) followed by a colon (:) when the CCP is waiting for console input. Thus, the sequence of prompts and commands shown below might occur after the CP/M system is loaded from disk A:

16K CP/M VER 1.4

A>DIR List all files on disk A.

SAMPLE ASM

SAMPLE PRN

A>B: Switch to disk B.

B>DIR *.ASM List all "ASM" files on B.

DUMP ASM

FILES ASM

B>A: Switch back to A.

4. THE FORM OF BUILT-IN COMMANDS.

The file and device reference forms described above can now be used to fully specify the structure of the built-in commands. In the description below, assume the following abbreviations:

ufn	-	unambiguous file reference
afn	-	ambiguous file reference
cr	-	carriage return

Further, recall that the CCP always translates lower case characters to upper case characters internally. Thus, lower case alphabets are treated as if they are upper case in command names and file references.

4.1 ERA afn cr

The ERA (erase) command removes files from the currently logged-in disk (i.e., the disk name currently prompted by CP/M preceding the ">"). The files which are erased are those which satisfy the ambiguous file reference afn. The following examples illustrate the use of ERA:

ERA X.Y	The file named X.Y on the currently logged disk is removed from the disk directory, and the space is returned.
ERA X.*	All files with primary name X are removed from the current disk.
ERA *.ASM	All files with secondary name ASM are removed from the current disk.
ERA X?Y.C?M	All files on the current disk which satisfy the ambiguous reference X?Y.C?M are deleted.
ERA *.*	Erase all files on the current disk (in this case the CCP prompts the console with the message "ALL FILES (Y/N)?" which requires a Y response before files are actually removed).
ERA B:*.PRN	All files on drive B which satisfy the ambiguous reference ???????.PRN are deleted, independently of the currently logged disk.

4.2. DIR afn cr

The DIR (directory) command causes the names of all files which satisfy the ambiguous file name afn to be listed at the console device. As a special case, the command

DIR

lists the files on the currently logged disk (the command "DIR" is equivalent to the command "DIR *.*"). Valid DIR commands are shown below.

DIR X.Y

DIR X?Z.C?M

DIR ??..Y

Similar to other CCP commands, the afn can be preceded by a drive name. The following DIR commands cause the selected drive to be addressed before the directory search takes place.

DIR B:

DIR B:X.Y

DIR B:*.A?M

If no files can be found on the selected diskette which satisfy the directory request, then the message "NOT FOUND" is typed at the console.

4.3. REN ufn1=ufn2 cr

The REN (rename) command allows the user to change the names of files on disk. The file satisfying ufn2 is changed to ufn1. The currently logged disk is assumed to contain the file to rename (ufn1). The CCP also allows the user to type a left-directed arrow instead of the equal sign, if the user's console supports this graphic character. Examples of the REN command are

REN X.Y=Q.R

The file Q.R is changed to X.Y.

REN XYZ.COM=XYZ.XXX

The file XYZ.XXX is changed to XYZ.COM.

The operator can precede either ufn1 or ufn2 (or both) by an optional drive address. Given that ufn1 is preceded by a drive name, then ufn2 is assumed to exist on the same drive as ufn1. Similarly, if ufn2 is preceded by a drive name, then ufn1 is assumed to reside on that drive as well. If both ufn1 and ufn2 are preceded by drive names, then the same drive must be

specified in both cases. The following REN commands illustrate this format.

REN A:X.ASM = Y.ASM	The file Y.ASM is changed to X.ASM on drive A.
REN B:ZAP.BAS=ZOT.BAS	The file ZOT.BAS is changed to ZAP.BAS on drive B.
REN B:A.ASM = B:A.BAK	The file A.BAK is renamed to A.ASM on drive B.

If the file ufn1 is already present, the REN command will respond with the error "FILE EXISTS" and not perform the change. If ufn2 does not exist on the specified diskette, then the message "NOT FOUND" is printed at the console.

4.4. SAVE n ufn cr

The SAVE command places n pages (256-byte blocks) onto disk from the TPA and names this file ufn. In the CP/M distribution system, the TPA starts at 100H (hexadecimal), which is the second page of memory. Thus, if the user's program occupies the area from 100H through 2FFH, the SAVE command must specify 2 pages of memory. The machine code file can be subsequently loaded and executed. Examples are:

SAVE 3 X.COM	Copies 100H through 3FFH to X.COM.
SAVE 40 Q	Copies 100H through 28FFH to Q (note that 28 is the page count in 28FFH, and that 28H = 2*16+8 = 40 decimal).
SAVE 4 X.Y	Copies 100H through 4FFH to X.Y.

The SAVE command can also specify a disk drive in the afn portion of the command, as shown below.

SAVE 10 B:ZOT.COM	Copies 10 pages (100H through 0AFFH) to the file ZOT.COM on drive B.
-------------------	--

4.5. TYPE ufn cr

The TYPE command displays the contents of the ASCII source file ufn on the currently logged disk at the console device. Valid TYPE commands are

TYPE X.Y

TYPE X.PLM

TYPE XXX

The TYPE command expands tabs (clt-I characters), assuming tab positions are set at every eighth column. The ufn can also reference a drive name as shown below.

TYPE B:X.PRN

The file X.PRN from drive B is displayed.

5. LINE EDITING AND OUTPUT CONTROL.

The CCP allows certain line editing functions while typing command lines.

rubout	Delete and echo the last character typed at the console.
ctl-U	Delete the entire line typed at the console.
ctl-X	(Same as ctl-U)
ctl-R	Retype current command line: types a "clean line" following character deletion with rubouts.
ctl-E	Physical end of line: carriage is returned, but line is not sent until the carriage return key is depressed.
ctl-C	CP/M system reboot (warm start)
ctl-Z	End input from the console (used in PIP and ED).

The control functions ctl-P and ctl-S affect console output as shown below.

ctl-P	Copy all subsequent console output to the currently assigned list device (see the STAT command). Output is sent to both the list device and the console device until the next ctl-P is typed.
ctl-S	Stop the console output temporarily. Program execution and output continue when the next character is typed at the console (e.g., another ctl-S). This feature is used to stop output on high speed consoles, such as CRT's, in order to view a segment of output before continuing.

Note that the ctl-key sequences shown above are obtained by depressing the control and letter keys simultaneously. Further, CCP command lines can generally be up to 255 characters in length; they are not acted upon until the carriage return key is typed.

6. TRANSIENT COMMANDS.

Transient commands are loaded from the currently logged disk and executed in the TPA. The transient commands defined for execution under the CCP are shown below. Additional functions can easily be defined by the user (see the LOAD command definition).

STAT	List the number of bytes of storage remaining on the currently logged disk, provide statistical information about particular files, and display or alter device assignment.
ASM	Load the CP/M assembler and assemble the specified program from disk.
LOAD	Load the file in Intel "hex" machine code format and produce a file in machine executable form which can be loaded into the TPA (this loaded program becomes a new command under the CCP).
DDT	Load the CP/M debugger into TPA and start execution.
PIP	Load the Peripheral Interchange Program for subsequent disk file and peripheral transfer operations.
ED	Load and execute the CP/M text editor program.
SYSGEN	Create a new CP/M system diskette.
SUBMIT	Submit a file of commands for batch processing.
DUMP	Dump the contents of a file in hex.
MOVCPM	Regenerate the CP/M system for a particular memory size.

Transient commands are specified in the same manner as built-in commands, and additional commands can be easily defined by the user. As an added convenience, the transient command can be preceded by a drive name, which causes the transient to be loaded from the specified drive into the TPA for execution. Thus, the command

B:STAT

causes CP/M to temporarily "log in" drive B for the source of the STAT transient, and then return to the original logged disk for subsequent processing.

The basic transient commands are listed in detail below.

6.1. STAT cr

The STAT command provides general statistical information about file storage and device assignment. It is initiated by typing one of the following forms:

```
STAT cr
STAT "command line" cr
```

Special forms of the "command line" allow the current device assignment to be examined and altered as well. The various command lines which can be specified are shown below, with an explanation of each form shown to the right.

STAT cr

If the user types an empty command line, the STAT transient calculates the storage remaining on all active drives, and prints a message

```
x: R/W, SPACE: nnnK
```

or

```
x: R/O, SPACE: nnnK
```

for each active drive x, where R/W indicates the drive may be read or written, and R/O indicates the drive is read only (a drive becomes R/O by explicitly setting it to read only, as shown below, or by inadvertently changing diskettes without performing a warm start). The space remaining on the diskette in drive x is given in kilobytes by nnn.

STAT x: cr

If a drive name is given, then the drive is selected before the storage is computed. Thus, the command "STAT B:" could be issued while logged into drive A, resulting in the message

```
BYTES REMAINING ON B: nnnK
```

STAT afn cr

The command line can also specify a set of files to be scanned by STAT. The files which satisfy afn are listed in alphabetical order, with storage requirements for each file under the heading

```
RECS BYTS EX D:FILENAME.TYP
rrrr bbbK ee d:pppppppp.sss
```

where rrrr is the number of 128-byte records

allocated to the file, bbb is the number of kilobytes allocated to the file ($bbb = rrrr * 128 / 1024$), ee is the number of 16K extensions ($ee = bbb / 16$), d is the drive name containing the file (A...Z), pppppppp is the (up to) eight-character primary file name, and sss is the (up to) three-character secondary name. After listing the individual files, the storage usage is summarized.

STAT x:afn cr

As a convenience, the drive name can be given ahead of the afn. In this case, the specified drive is first selected, and the form "STAT afn" is executed.

STAT x:=R/O cr

This form sets the drive given by x to read-only, which remains in effect until the next warm or cold start takes place. When a disk is read-only, the message

BDOS ERR ON x: READ ONLY

will appear if there is an attempt to write to the read-only disk x. CP/M waits until a key is depressed before performing an automatic warm start (at which time the disk becomes R/W).

The STAT command also allows control over the physical to logical device assignment (see the IOBYTE function described in the manuals "CP/M Interface Guide" and "CP/M System Alteration Guide"). In general, there are four logical peripheral devices which are, at any particular instant, each assigned to one of several physical peripheral devices. The four logical devices are named:

CON:	The system console device (used by CCP for communication with the operator)
RDR:	The paper tape reader device
PUN:	The paper tape punch device
LST:	The output list device

The actual devices attached to any particular computer system are driven by subroutines in the BIOS portion of CP/M. Thus, the logical RDR: device, for example, could actually be a high speed reader, Teletype reader, or cassette tape. In order to allow some flexibility in device naming and assignment, several physical devices are defined, as shown below:

TTY:	Teletype device (slow speed console)
CRT:	Cathode ray tube device (high speed console)
BAT:	Batch processing (console is current RDR:, output goes to current LST: device)
UC1:	User-defined console
PTR:	Paper tape reader (high speed reader)
UR1:	User-defined reader #1
UR2:	User-defined reader #2
PTP:	Paper tape punch (high speed punch)
UP1:	User-defined punch #1
UP2:	User-defined punch #2
LPT:	Line printer
UL1:	User-defined list device #1

It must be emphasized that the physical device names may or may not actually correspond to devices which the names imply. That is, the PTP: device may be implemented as a cassette write operation, if the user wishes. The exact correspondence and driving subroutine is defined in the BIOS portion of CP/M. In the standard distribution version of CP/M, these devices correspond to their names on the MDS 800 development system.

The possible logical to physical device assignments can be displayed by typing

```
STAT VAL: cr
```

The STAT prints the possible values which can be taken on for each logical device:

```
CON. = TTY: CRT: BAT: UC1:
RDR: = TTY: PTR: UR1: UR2:
PUN: = TTY: PTP: UP1: UP2:
LST: = TTY: CRT: LPT: UL1:
```

In each case, the logical device shown to the left can take any of the four physical assignments shown to the right on each line. The current logical to physical mapping is displayed by typing the command

```
STAT DEV: cr
```

which produces a listing of each logical device to the left, and the current corresponding physical device to the right. For example, the list might appear as follows:

```
CON: = CRT:
RDR: = URL:
PUN: = PTP:
LST: = TTY:
```

The current logical to physical device assignment can be changed by typing a STAT command of the form

```
STAT ld1 = pd1, ld2 = pd2 , ... , ldn = pdn cr
```

where ld1 through ldn are logical device names, and pd1 through pdn are compatible physical device names (i.e., ldi and pdi appear on the same line in the "VAL:" command shown above). The following are valid STAT commands which change the current logical to physical device assignments:

```
STAT CON:=CRT: cr
STAT PUN: = TTY:,LST:=LPT:, RDR:=TTY: cr
```

6.2. ASM ufn cr

The ASM command loads and executes the CP/M 8080 assembler. The ufn specifies a source file containing assembly language statements where the secondary name is assumed to be ASM, and thus is not specified. The following ASM commands are valid:

```
ASM X
```

```
ASM GAMMA
```

The two-pass assembler is automatically executed. If assembly errors occur during the second pass, the errors are printed at the console.

The assembler produces a file

```
x.PRN
```

where x is the primary name specified in the ASM command. The PRN file contains a listing of the source program (with imbedded tab characters if present in the source program), along with the machine code generated for each statement and diagnostic error messages, if any. The PRN file can be listed

at the console using the TYPE command, or sent to a peripheral device using PIP (see the PIP command structure below). Note also that the PRN file contains the original source program, augmented by miscellaneous assembly information in the leftmost 16 columns (program addresses and hexadecimal machine code, for example). Thus, the PRN file can serve as a backup for the original source file: if the source file is accidentally removed or destroyed, the PRN file can be edited (see the ED operator's guide) by removing the leftmost 16 characters of each line (this can be done by issuing a single editor "macro" command). The resulting file is identical to the original source file and can be renamed (REN) from PRN to ASM for subsequent editing and assembly. The file

x.HEX

is also produced which contains 8080 machine language in Intel "hex" format suitable for subsequent loading and execution (see the LOAD command). For complete details of CP/M's assembly language program, see the "CP/M Assembler Language (ASM) User's Guide."

Similar to other transient commands, the source file for assembly can be taken from an alternate disk by prefixing the assembly language file name by a disk drive name. Thus, the command

ASM B:ALPHA cr

loads the assembler from the currently logged drive and operates upon the source program ALPHA.ASM on drive B. The HEX and PRN files are also placed on drive B in this case.

6.3. LOAD ufn cr

The LOAD command reads the file ufn, which is assumed to contain "hex" format machine code, and produces a memory image file which can be subsequently executed. The file name ufn is assumed to be of the form

x.HEX

and thus only the name x need be specified in the command. The LOAD command creates a file named

x.COM

which marks it as containing machine executable code. The file is actually loaded into memory and executed when the user types the file name x immediately after the prompting character ">" printed by the CCP.

In general, the CCP reads the name x following the prompting character and looks for a built-in function name. If no function name is found, the CCP searches the system disk directory for a file by the name

x.COM

If found, the machine code is loaded into the TPA, and the program executes. Thus, the user need only LOAD a hex file once; it can be subsequently executed any number of times by simply typing the primary name. In this way, the user can "invent" new commands in the CCP. (Initialized disks contain the transient commands as COM files, which can be deleted at the user's option.) The operation can take place on an alternate drive if the file name is prefixed by a drive name. Thus,

LOAD B:BETA

brings the LOAD program into the TPA from the currently logged disk and operates upon drive B after execution begins.

It must be noted that the BETA.HEX file must contain valid Intel format hexadecimal machine code records (as produced by the ASM program, for example) which begin at 100H, the beginning of the TPA. Further, the addresses in the hex records must be in ascending order; gaps in unfilled memory regions are filled with zeroes by the LOAD command as the hex records are read. Thus, LOAD must be used only for creating CP/M standard "COM" files which operate in the TPA. Programs which occupy regions of memory other than the TPA can be loaded under DDT.

6.4. PIP cr

PIP is the CP/M Peripheral Interchange Program which implements the basic media conversion operations necessary to load, print, punch, copy, and combine disk files. The PIP program is initiated by typing one of the following forms

- (1) PIP cr
- (2) PIP "command line" cr

In both cases, PIP is loaded into the TPA and executed. In case (1), PIP reads command lines directly from the console, prompted with the "*" character, until an empty command line is typed (i.e., a single carriage return is issued by the operator). Each successive command line causes some media conversion to take place according to the rules shown below. Form (2) of the PIP command is equivalent to the first, except that the single command line given with the PIP command is automatically executed, and PIP terminates immediately with no further prompting of the console for input command lines. The form of each command line is

destination = source#1, source#2, ... , source#n cr

where "destination" is the file or peripheral device to receive the data, and

"source#1, ..., source#n" represents a series of one or more files or devices which are copied from left to right to the destination.

When multiple files are given in the command line (i.e, $n > 1$), the individual files are assumed to contain ASCII characters, with an assumed CP/M end-of-file character (ctl-Z) at the end of each file (see the O parameter to override this assumption). The equal symbol (=) can be replaced by a left-oriented arrow, if your console supports this ASCII character, to improve readability. Lower case ASCII alphabets are internally translated to upper case to be consistent with CP/M file and device name conventions. Finally, the total command line length cannot exceed 255 characters (ctl-E can be used to force a physical carriage return for lines which exceed the console width).

The destination and source elements can be unambiguous references to CP/M source files, with or without a preceding disk drive name. That is, any file can be referenced with a preceding drive name (A:, B:, C:, or D:) which defines the particular drive where the file may be obtained or stored. When the drive name is not included, the currently logged disk is assumed. Further, the destination file can also appear as one or more of the source files, in which case the source file is not altered until the entire concatenation is complete. If the destination file already exists, it is removed if the command line is properly formed (it is not removed if an error condition arises). The following command lines (with explanations to the right) are valid as input to PIP:

X = Y cr

Copy to file X from file Y, where X and Y are unambiguous file names; Y remains unchanged.

X = Y,Z cr

Concatenate files Y and Z and copy to file X, with Y and Z unchanged.

X.ASM=Y.ASM,Z.ASM,FIN.ASM cr

Create the file X.ASM from the concatenation of the Y, Z, and FIN files with type ASM.

NEW.ZOT = B:OLD.ZAP cr

Move a copy of OLD.ZAP from drive B to the currently logged disk; name the file NEW.ZOT.

B:A.U = B:B.V,A:C.W,D:X cr

Concatenate file B.V from drive B with C.W from drive A and D.X from the logged disk; create the file A.U on drive B.

For more convenient use, PIP allows abbreviated commands for transferring files between disk drives. The abbreviated forms are

PIP x:=afn cr

PIP x:=y:afn cr

PIP ufn = y: cr

PIP x:ufn = y: cr

The first form copies all files from the currently logged disk which satisfy the afn to the same file names on drive x (x = A...Z). The second form is equivalent to the first, where the source for the copy is drive y (y = A...Z). The third form is equivalent to the command "PIP ufn=y:ufn cr" which copies the file given by ufn from drive y to the file ufn on drive x. The fourth form is equivalent to the third, where the source disk is explicitly given by y.

Note that the source and destination disks must be different in all of these cases. If an afn is specified, PIP lists each ufn which satisfies the afn as it is being copied. If a file exists by the same name as the destination file, it is removed upon successful completion of the copy, and replaced by the copied file.

The following PIP commands give examples of valid disk-to-disk copy operations:

B:=*.COM cr

Copy all files which have the secondary name "COM" to drive B from the current drive.

A:=B:ZAP.* cr

Copy all files which have the primary name "ZAP" to drive A from drive B.

ZAP.ASM=B: cr

Equivalent to ZAP.ASM=B:ZAP.ASM

B:ZOT.COM=A: cr

Equivalent to B:ZOT.COM=A:ZOT.COM

B:=GAMMA.BAS cr

Same as B:GAMMA.BAS=GAMMA.BAS

B:=A:GAMMA.BAS cr

Same as B:GAMMA.BAS=A:GAMMA.BAS

PIP also allows reference to physical and logical devices which are attached to the CP/M system. The device names are the same as given under the STAT command, along with a number of specially named devices. The logical devices given in the STAT command are

CON: (console), RDR: (reader), PUN: (punch), and LST: (list)

while the physical devices are

TTY: (console, reader, punch, or list)
 CRT: (console, or list), UC1: (console)
 PTR: (reader), UR1: (reader), UR2: (reader)
 PTP: (punch), UP1: (punch), UP2: (punch)
 LPT: (list), ULL: (list)

(Note that the "BAT:" physical device is not included, since this assignment is used only to indicate that the RDR: and LST: devices are to be used for console input/output.)

The RDR, LST, PUN, and CON devices are all defined within the BIOS portion of CP/M, and thus are easily altered for any particular I/O system. (The current physical device mapping is defined by IOBYTE; see the "CP/M Interface Guide" for a discussion of this function). The destination device must be capable of receiving data (i.e., data cannot be sent to the punch), and the source devices must be capable of generating data (i.e., the LST: device cannot be read).

The additional device names which can be used in PIP commands are

NUL: Send 40 "nulls" (ASCII 0's) to the device
 (this can be issued at the end of punched output).

EOF: Send a CP/M end-of-file (ASCII ctl-Z) to the
 destination device (sent automatically at the
 end of all ASCII data transfers through PIP).

INP: Special PIP input source which can be "patched"
 into the PIP program itself: PIP gets the input
 data character-by-character by CALLing location
 103H, with data returned in location 109H (parity
 bit must be zero).

OUT: Special PIP output destination which can be
 patched into the PIP program: PIP CALLs location
 106H with data in register C for each character
 to transmit. Note that locations 109H through
 1FFH of the PIP memory image are not used and
 can be replaced by special purpose drivers using
 DDT (see the DDT operator's manual).

PRN: Same as LST:, except that tabs are expanded at
 every eighth character position, lines are
 numbered, and page ejects are inserted every 60
 lines, with an initial eject (same as [t8np]).

File and device names can be interspersed in the PIP commands. In each case, the specific device is read until end-of-file (ctl-Z for ASCII files, and a real end of file for non-ASCII disk files). Data from each device or file is concatenated from left to right until the last data source has been

read. The destination device or file is written using the data from the source files, and an end-of-file character (ctl-Z) is appended to the result for ASCII files. Note if the destination is a disk file, then a temporary file is created (\$\$\$ secondary name) which is changed to the actual file name only upon successful completion of the copy. Files with the extension "COM" are always assumed to be non-ASCII.

The copy operation can be aborted at any time by depressing any key on the keyboard (a rubout suffices). PIP will respond with the message "ABORTED" to indicate that the operation was not completed. Note that if any operation is aborted, or if an error occurs during processing, PIP removes any pending commands which were set up while using the SUBMIT command.

It should also be noted that PIP performs a special function if the destination is a disk file with type "HEX" (an Intel hex formatted machine code file), and the source is an external peripheral device, such as a paper tape reader. In this case, the PIP program checks to ensure that the source file contains a properly formed hex file, with legal hexadecimal values and checksum records. When an invalid input record is found, PIP reports an error message at the console and waits for corrective action. It is usually sufficient to open the reader and rerun a section of the tape (pull the tape back about 20 inches). When the tape is ready for the re-read, type a single carriage return at the console, and PIP will attempt another read. If the tape position cannot be properly read, simply continue the read (by typing a return following the error message), and enter the record manually with the ED program after the disk file is constructed. For convenience, PIP allows the end-of-file to be entered from the console if the source file is a RDR: device. In this case, the PIP program reads the device and monitors the keyboard. If ctl-Z is typed at the keyboard, then the read operation is terminated normally.

Valid PIP commands are shown below.

PIP LST: = X.PRN cr

Copy X.PRN to the LST device and terminate the PIP program.

PIP cr

Start PIP for a sequence of commands (PIP prompts with "*").

*CON:=X.ASM,Y.ASM,Z.ASM cr

Concatenate three ASM files and copy to the CON device.

*X.HEX=CON:,Y.HEX,PTR: cr

Create a HEX file by reading the CON (until a ctl-Z is typed), followed by data from Y.HEX, followed by data from PTR until a ctl-Z is encountered.

*cr

Single carriage return stops PIP.

PIP PUN:=NUL: ,X.ASM,EOF: ,NUL: cr

Send 40 nulls to the punch device; then copy the X.ASM file to the punch, followed by an end-of-file (ctl-Z) and 40 more null characters.

The user can also specify one or more PIP parameters, enclosed in left and right square brackets, separated by zero or more blanks. Each parameter affects the copy operation, and the enclosed list of parameters must immediately follow the affected file or device. Generally, each parameter can be followed by an optional decimal integer value (the S and Q parameters are exceptions). The valid PIP parameters are listed below.

- B Block mode transfer: data is buffered by PIP until an ASCII x-off character (ctl-S) is received from the source device. This allows transfer of data to a disk file from a continuous reading device, such as a cassette reader. Upon receipt of the x-off, PIP clears the disk buffers and returns for more input data. The amount of data which can be buffered is dependent upon the memory size of the host system (PIP will issue an error message if the buffers overflow).
- Dn Delete characters which extend past column n in the transfer of data to the destination from the character source. This parameter is used most often to truncate long lines which are sent to a (narrow) printer or console device.
- E Echo all transfer operations to the console as they are being performed.
- F Filter form feeds from the file. All imbedded form feeds are removed. The P parameter can be used simultaneously to insert new form feeds.
- H Hex data transfer: all data is checked for proper Intel hex file format. Non-essential characters between hex records are removed during the copy operation. The console will be prompted for corrective action in case errors occur.
- I Ignore ":00" records in the transfer of Intel hex format file (the I parameter automatically sets the H parameter).
- L Translate upper case alphabets to lower case.
- N Add line numbers to each line transferred to the destination starting at one, and incrementing by 1. Leading zeroes are suppressed, and the number is followed by a colon. If N2 is specified, then leading zeroes are included, and a tab is inserted following the number. The tab is expanded if T is

set.

- O Object file (non-ASCII) transfer: the normal CP/M end of file is ignored.
- Pn Include page ejects at every n lines (with an initial page eject). If n = 1 or is excluded altogether, page ejects occur every 60 lines. If the F parameter is used, form feed suppression takes place before the new page ejects are inserted.
- Qs↑z Quit copying from the source device or file when the string s (terminated by ctl-Z) is encountered.
- Ss↑z Start copying from the source device when the string s is encountered (terminated by ctl-Z). The S and Q parameters can be used to "abstract" a particular section of a file (such as a subroutine). The start and quit strings are always included in the copy operation.

NOTE - the strings following the s and q parameters are translated to upper case by the CCP if form (2) of the PIP command is used. Form (1) of the PIP invocation, however, does not perform the automatic upper case translation.

- (1) PIP cr
- (2) PIP "command line" cr

- Tn Expand tabs (ctl-I characters) to every nth column during the transfer of characters to the destination from the source.
- U Translate lower case alphabets to upper case during the the copy operation.
- V Verify that data has been copied correctly by rereading after the write operation (the destination must be a disk file).
- Z Zero the parity bit on input for each ASCII character.

The following are valid PIP commands which specify parameters in the file transfer:

- PIP X.ASM=B:[v] cr Copy X.ASM from drive B to the current drive and verify that the data was properly copied.
- PIP LPT:=X.ASM[nt8u] cr Copy X.ASM to the LPT: device; number each line, expand tabs to every eighth column, and translate lower case alphabets to upper case.

PIP PUN:=X.HEX[i],Y.ZOT[h] cr First copy X.HEX to the PUN: device and ignore the trailing ":00" record in X.HEX; then continue the transfer of data by reading Y.ZOT, which contains hex records, including any ":00" records which it contains.

PIP X.LIB = Y.ASM [sSUBR1:↑z qJMP L3↑z] cr Copy from the file Y.ASM into the file X.LIB. Start the copy when the string "SUBR1:" has been found, and quit copying after the string "JMP L3" is encountered.

PIP PRN:=X.ASM[p50] Send X.ASM to the LST: device, with line numbers, tabs expanded to every eighth column, and page ejects at every 50th line. Note that nt8p60 is the assumed parameter list for a PRN file; p50 overrides the default value.

6.5. ED ufn cr

The ED program is the CP/M system context editor, which allows creation and alteration of ASCII files in the CP/M environment. Complete details of operation are given in the ED user's manual, "ED: a Context Editor for the CP/M Disk System." In general, ED allows the operator to create and operate upon source files which are organized as a sequence of ASCII characters, separated by end-of-line characters (a carriage-return line-feed sequence). There is no practical restriction on line length (no single line can exceed the size of the working memory), which is instead defined by the number of characters typed between cr's. The ED program has a number of commands for character string searching, replacement, and insertion, which are useful in the creation and correction of programs or text files under CP/M. Although the CP/M has a limited memory work space area (approximately 5000 characters in a 16K CP/M system), the file size which can be edited is not limited, since data is easily "paged" through this work area.

Upon initiation, ED creates the specified source file, if it does not exist, and opens the file for access. The programmer then "appends" data from the source file into the work area, if the source file already exists (see the A command), for editing. The appended data can then be displayed, altered, and written from the work area back to the disk (see the W command). Particular points in the program can be automatically paged and located by context (see the N command), allowing easy access to particular portions of a large file.

Given that the operator has typed

ED X.ASM cr

the ED program creates an intermediate work file with the name

X.\$\$\$

to hold the edited data during the ED run. Upon completion of ED, the X.ASM file (original file) is renamed to X.BAK, and the edited work file is renamed to X.ASM. Thus, the X.BAK file contains the original (unedited) file, and the X.ASM file contains the newly edited file. The operator can always return to the previous version of a file by removing the most recent version, and renaming the previous version. Suppose, for example, that the current X.ASM file was improperly edited; the sequence of CCP command shown below would reclaim the backup file.

DIR X.*	Check to see that BAK file is available.
ERA X.ASM	Erase most recent version.
REN X.ASM=X.BAK	Rename the BAK file to ASM.

Note that the operator can abort the edit at any point (reboot, power failure, ctl-C, or Q command) without destroying the original file. In this case, the BAK file is not created, and the original file is always intact.

The ED program also allows the user to "ping-pong" the source and create backup files between two disks. The form of the ED command in this case is

ED ufn d:

where ufn is the name of a file to edit on the currently logged disk, and d is the name of an alternate drive. The ED program reads and processes the source file, and writes the new file to drive d, using the name ufn. Upon completion of processing, the original file becomes the backup file. Thus, if the operator is addressing disk A, the following command is valid:

ED X.ASM B:

which edits the file X.ASM on drive A, creating the new file X.\$\$\$ on drive B. Upon completion of a successful edit, A:X.ASM is renamed to A:X.BAK, and B:X.\$\$\$ is renamed to B:X.ASM. For user convenience, the currently logged disk becomes drive B at the end of the edit. Note that if a file by the name B:X.ASM exists before the editing begins, the message

FILE EXISTS

is printed at the console as a precaution against accidentally destroying a source file. In this case, the operator must first ERASE the existing file and then restart the edit operation.

Similar to other transient commands, editing can take place on a drive different from the currently logged disk by preceding the source file name by a drive name. Examples of valid edit requests are shown below

ED A:X.ASM

Edit the file X.ASM on drive A, with new file and backup on drive A.

ED B:X.ASM A:

Edit the file X.ASM on drive B to the temporary file X.\$\$\$ on drive A. On termination of editing, change X.ASM on drive B to X.BAK, and change X.\$\$\$ on drive A to X.ASM.

6.6. SYSGEN cr

The SYSGEN transient command allows generation of an initialized diskette containing the CP/M operating system. The SYSGEN program prompts the console for commands, with interaction as shown below.

SYSGEN cr

Initiate the SYSGEN program.

SYSGEN VERSION m.m

SYSGEN sign-on message.

SOURCE DRIVE NAME (OR RETURN TO SKIP)

Respond with the drive name (one of the letters A, B, C, or D) of the disk containing a CP/M system; usually A. If a copy of CP/M already exists in memory, due to a MOVCPM command, type a cr only. Typing a drive name x will cause the response:

SOURCE ON x THEN TYPE RETURN

Place a diskette containing the CP/M operating system on drive x (x is one of A, B, C, or D). Answer with cr when ready.

FUNCTION COMPLETE

System is copied to memory. SYSGEN will then prompt with:

DESTINATION DRIVE NAME (OR RETURN TO REBOOT)

If a diskette is being initialized, place the new disk into a drive and answer with the drive name. Otherwise, type a cr and the system will reboot from drive A. Typing drive name x will cause SYSGEN to prompt

with:

DESTINATION ON x THEN TYPE RETURN Place new diskette into drive
x; type return when ready.

FUNCTION COMPLETE

New diskette is initialized
in drive x.

The "DESTINATION" prompt will be repeated until a single carriage return is typed at the console, so that more than one disk can be initialized.

Upon completion of a successful system generation, the new diskette contains the operating system, and only the built-in commands are available. A factory-fresh IBM-compatible diskette appears to CP/M as a diskette with an empty directory; therefore, the operator must copy the appropriate COM files from an existing CP/M diskette to the newly constructed diskette using the PIP transient.

The user can copy all files from an existing diskette by typing the PIP command

PIP B: = A: *.*[v] cr

which copies all files from disk drive A to disk drive B, and verifies that each file has been copied correctly. The name of each file is displayed at the console as the copy operation proceeds.

It should be noted that a SYSGEN does not destroy the files which already exist on a diskette; it results only in construction of a new operating system. Further, if a diskette is being used only on drives B through D, and will never be the source of a bootstrap operation on drive A, the SYSGEN need not take place. In fact, a new diskette needs absolutely no initialization to be used with CP/M.

6.7. SUBMIT ufn parm#1 ... parm#n cr

The SUBMIT command allows CP/M commands to be batched together for automatic processing. The ufn given in the SUBMIT command must be the filename of a file which exists on the currently logged disk, with an assumed file type of "SUB." The SUB file contains CP/M prototype commands, with possible parameter substitution. The actual parameters parm#1 ... parm#n are substituted into the prototype commands, and, if no errors occur, the file of substituted commands are processed sequentially by CP/M.

The prototype command file is created using the ED program, with interspersed "\$" parameters of the form

\$1 \$2 \$3 ... \$n

corresponding to the number of actual parameters which will be included when the file is submitted for execution. When the SUBMIT transient is executed, the actual parameters parm#1 ... parm#n are paired with the formal parameters \$1 ... \$n in the prototype commands. If the number of formal and actual parameters does not correspond, then the submit function is aborted with an error message at the console. The SUBMIT function creates a file of substituted commands with the name

\$\$\$SUB

on the logged disk. When the system reboots (at the termination of the SUBMIT), this command file is read by the CCP as a source of input, rather than the console. If the SUBMIT function is performed on any disk other than drive A, the commands are not processed until the disk is inserted into drive A and the system reboots. Further, the user can abort command processing at any time by typing a rubout when the command is read and echoed. In this case, the \$\$\$SUB file is removed, and the subsequent commands come from the console. Command processing is also aborted if the CCP detects an error in any of the commands. Programs which execute under CP/M can abort processing of command files when error conditions occur by simply erasing any existing \$\$\$SUB file.

In order to introduce dollar signs into a SUBMIT file, the user may type a "\$\$" which reduces to a single "\$" within the command file. Further, an up-arrow symbol "↑" may precede an alphabetic character x, which produces a single ctl-x character within the file.

The last command in a SUB file can initiate another SUB file, thus allowing chained batch commands.

Suppose the file ASMBL.SUB exists on disk and contains the prototype commands

```
ASM $1
DIR $1.*
ERA *.BAK
PIP $2:=$1.PRN
ERA $1.PRN
```

and the command

SUBMIT ASMBL X PRN cr

is issued by the operator. The SUBMIT program reads the ASMBL.SUB file, substituting "X" for all occurrences of \$1 and "PRN" for all occurrences of \$2, resulting in a \$\$\$SUB file containing the commands

```

ASM X
DIR X.*
ERA *.BAK
PIP PRN:=X.PRN
ERA X.PRN

```

which are executed in sequence by the CCP.

The SUBMIT function can access a SUB file which is on an alternate drive by preceding the file name by a drive name. Submitted files are only acted upon, however, when they appear on drive A. Thus, it is possible to create a submitted file on drive B which is executed at a later time when it is inserted in drive A.

6.8. DUMP ufn cr

The DUMP program types the contents of the disk file (ufn) at the console in hexadecimal form. The file contents are listed sixteen bytes at a time, with the absolute byte address listed to the left of each line in hexadecimal. Long typeouts can be aborted by pushing the rubout key during printout. (The source listing of the DUMP program is given in the "CP/M Interface Guide" as an example of a program written for the CP/M environment.)

6.9. MOVCPM cr

The MOVCPM program allows the user to reconfigure the CP/M system for any particular memory size. Two optional parameters may be used to indicate (1) the desired size of the new system and (2) the disposition of the new system at program termination. If the first parameter is omitted or a "*" is given, the MOVCPM program will reconfigure the system to its maximum size, based upon the kilobytes of contiguous RAM in the host system (starting at 0000H). If the second parameter is omitted, the system is executed, but not permanently recorded; if "*" is given, the system is left in memory, ready for a SYSGEN operation. The MOVCPM program relocates a memory image of CP/M and places this image in memory in preparation for a system generation operation. The command forms are:

```
MOVCPM cr
```

Relocate and execute CP/M for management of the current memory configuration (memory is examined for contiguous RAM, starting at 100H). Upon completion of the relocation, the new system is executed but not permanently recorded on the diskette. The system which is constructed contains a BIOS for the Intel MDS 800.

MOVCPM n cr

Create a relocated CP/M system for management of an n kilobyte system (n must be in the range 16 to 64), and execute the system, as described above.

MOVCPM * * cr

Construct a relocated memory image for the current memory configuration, but leave the memory image in memory, in preparation for a SYSGEN operation.

MOVCPM n * cr

Construct a relocated memory image for an n kilobyte memory system, and leave the memory image in preparation for a SYSGEN operation.

The command

MOVCPM * *

for example, constructs a new version of the CP/M system and leaves it in memory, ready for a SYSGEN operation. The message

READY FOR "SYSGEN" OR
"SAVE 32 CPMxx.COM"

is printed at the console upon completion, where xx is the current memory size in kilobytes. The operator can then type

SYSGEN cr

Start the system generation.

SOURCE DRIVE NAME (OR RETURN TO SKIP) Respond with a cr to skip the CP/M read operation since the system is already in memory as a result of the previous MOVCPM operation.

DESTINATION DRIVE NAME (OR RETURN TO REBOOT)
Respond with B to write new system to the diskette in drive B. SYSGEN will prompt with:

DESTINATION ON B, THEN TYPE RETURN
Ready the fresh diskette on drive B and type a return when ready.

Note that if you respond with "A" rather than "B" above, the system will be written to drive A rather than B. SYSGEN will continue to type the prompt:

DESTINATION DRIVE NAME (OR RETURN TO REBOOT)

until the operator responds with a single carriage return, which stops the

SYSGEN program with a system reboot.

The user can then go through the reboot process with the old or new diskette. Instead of performing the SYSGEN operation, the user could have typed

SAVE 32 CPMxx.COM

at the completion of the MOVCP function, which would place the CP/M memory image on the currently logged disk in a form which can be "patched". This is necessary when operating in a non-standard environment where the BIOS must be altered for a particular peripheral device configuration, as described in the "CP/M System Alteration Guide".

Valid MOVCP commands are given below:

MOVCPM 48 cr	Construct a 48K version of CP/M and start execution
MOVCPM 48 * cr	Construct a 48K version of CP/M in preparation for permanent recording; response is
	READY FOR "SYSGEN" OR "SAVE 32 CPM48.COM"
MOVCPM * * CR	Construct a maximum memory version of CP/M and start execution.

It is important to note that the newly created system is serialized with the number attached to the original diskette and is subject to the conditions of the Exidy Inc. Software Licensing Agreement.

7. BDOS ERROR MESSAGES.

There are three error situations which the Basic Disk Operating System intercepts during file processing. When one of these conditions is detected, the BDOS prints the message:

BDOS ERR ON x: error

where x is the drive name, and "error" is one of the three error messages:

BAD SECTOR
SELECT
READ ONLY

The "BAD SECTOR" message indicates that the disk controller electronics has detected an error condition in reading or writing the diskette. This condition is generally due to a malfunctioning disk controller, or an extremely worn diskette. If you find that your system reports this error more than once a month, you should check the state of your controller electronics, and the condition of your media. You may also encounter this condition in reading files generated by a controller produced by a different manufacturer. Even though controllers are claimed to be IBM-compatible, one often finds small differences in recording formats. The MDS-800 controller, for example, requires two bytes of one's following the data CRC byte, which is not required in the IBM format. As a result, diskettes generated by the Intel MDS can be read by almost all other IBM-compatible systems, while disk files generated on other manufacturer's equipment will produce the "BAD SECTOR" message when read by the MDS. In any case, recovery from this condition is accomplished by typing a ctl-C to reboot (this is the safest!), or a return, which simply ignores the bad sector in the file operation. Note, however, that typing a return may destroy your diskette integrity if the operation is a directory write, so make sure you have adequate backups in this case.

The "SELECT" error occurs when there is an attempt to address a drive beyond the A through D range. In this case, the value of x in the error message gives the selected drive. The system reboots following any input from the console.

The "READ ONLY" message occurs when there is an attempt to write to a diskette which has been designated as read-only in a STAT command, or has been set to read-only by the BDOS. In general, the operator should reboot CP/M either by using the warm start procedure (ctl-C) or by performing a cold start whenever the diskettes are changed. If a changed diskette is to be read but not written, BDOS allows the diskette to be changed without the warm or cold start, but internally marks the drive as read-only. The status of the drive is subsequently changed to read/write if a warm or cold start occurs. Upon issuing this message, CP/M waits for input from the console. An automatic warm start takes place following any input.

8. OPERATION OF CP/M ON THE MDS.

This section gives operating procedures for using CP/M on the Intel MDS microcomputer development system. A basic knowledge of the MDS hardware and software systems is assumed.

CP/M is initiated in essentially the same manner as Intel's ISIS operating system. The disk drives are labelled 0 through 3 on the MDS, corresponding to CP/M drives A through D, respectively. The CP/M system diskette is inserted into drive 0, and the BOOT and RESET switches are depressed in sequence. The interrupt 2 light should go on at this point. The space bar is then depressed on the device which is to be taken as the system console, and the light should go out (if it does not, then check connections and baud rates). The BOOT switch is then turned off, and the CP/M signon message should appear at the selected console device, followed by the "A>" system prompt. The user can then issue the various resident and transient commands

The CP/M system can be restarted (warm start) at any time by pushing the INT 0 switch on the front panel. The built-in Intel ROM monitor can be initiated by pushing the INT 7 switch (which generates a RST 7), except when operating under DDT, in which case the DDT program gets control instead.

Diskettes can be removed from the drives at any time, and the system can be shut down during operation without affecting data integrity. Note, however, that the user must not remove a diskette and replace it with another without rebooting the system (cold or warm start), unless the inserted diskette is "read only."

Due to hardware hang-ups or malfunctions, CP/M may type the message

BDOS ERR ON x: BAD SECTOR

where x is the drive which has a permanent error. This error may occur when drive doors are opened and closed randomly, followed by disk operations, or may be due to a diskette, drive, or controller failure. The user can optionally elect to ignore the error by typing a single return at the console. The error may produce a bad data record, requiring re-initialization of up to 128 bytes of data. The operator can reboot the CP/M system and try the operation again.

Termination of a CP/M session requires no special action, except that it is necessary to remove the diskettes before turning the power off, to avoid random transients which often make their way to the drive electronics.

It should be noted that factory-fresh IBM-compatible diskettes should be used rather than diskettes which have previously been used with any ISIS version. In particular, the ISIS "FORMAT" operation produces non-standard sector numbering throughout the diskette. This non-standard numbering seriously degrades the performance of CP/M, and will operate noticeably slower

than the distribution version. If it becomes necessary to reformat a diskette (which should not be the case for standard diskettes), a program can be written under CP/M which causes the MDS 800 controller to reformat with sequential sector numbering (1-26) on each track.

Note: "MDS 800" and "ISIS" are registered trademarks of Intel Corporation.

CP/M INTERFACE GUIDE

1-11-66

TABLE OF CONTENTS

1.	INTRODUCTION	1
1.1	CP/M Organization	1
1.2	Operation of Transient Programs	1
1.3	Operating System Facilities	3
2.	BASIC I/O FACILITIES	4
2.1	Direct and Buffered I/O	5
2.2	A Simple Example	5
3.	DISK I/O FACILITIES	9
3.1	File System Organization	9
3.2	File Control Block Format	10
3.3	Disk Access Primitives	12
3.4	Random Access	18
4.	SYSTEM GENERATION	18
4.1	Initializing CP/M from an Existing Diskette	19
5.	CP/M ENTRY POINT SUMMARY	20
6.	ADDRESS ASSIGNMENTS	22
7.	SAMPLE PROGRAMS	23

1890-1891

1890-1891

1890-1891

1890-1891

1890-1891

1890-1891

1890-1891

1890-1891

1890-1891

1890-1891

1890-1891

1890-1891

1890-1891

1890-1891

1890-1891

1890-1891

1890-1891

1890-1891

1890-1891

1890-1891

1890-1891

1890-1891

1890-1891

1890-1891

1890-1891

1890-1891

1890-1891

1890-1891

CP/M INTERFACE GUIDE

1. INTRODUCTION

This manual describes the CP/M system organization including the structure of memory, as well as system entry points. The intention here is to provide the necessary information required to write programs which operate under CP/M, and which use the peripheral and disk I/O facilities of the system.

1.1 CP/M Organization

CP/M is logically divided into four parts:

- BIOS - the basic I/O system for serial peripheral control
- BDOS - the basic disk operating system primitives
- CCP - the console command processor
- TPA - the transient program area

The BIOS and BDOS are combined into a single program with a common entry point and referred to as the FDOS. The CCP is a distinct program which uses the FDOS to provide a human-oriented interface to the information which is cataloged on the diskette. The TPA is an area of memory (i.e., the portion which is not used by the FDOS and CCP) where various non-resident operating system commands are executed. User programs also execute in the TPA. The organization of memory in a standard CP/M system is shown in Figure 1.

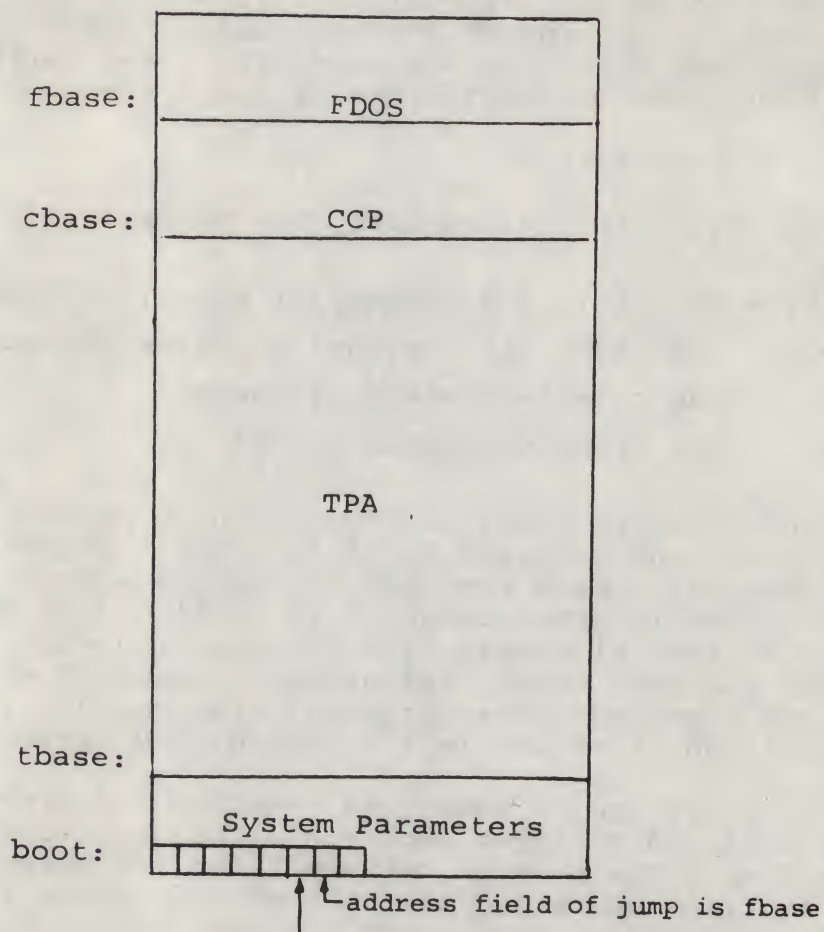
The lower portion of memory is reserved for system information (which is detailed in later sections), including user defined interrupt locations. The portion between tbase and cbase is reserved for the transient operating system commands, while the portion above cbase contains the resident CCP and FDOS. The last three locations of memory contain a jump instruction to the FDOS entry point which provides access to system functions.

1.2 Operation of Transient Programs

Transient programs (system functions and user-defined programs) are loaded into the TPA and executed as follows. The operator communicates with the CCP by typing command lines following each prompt character. Each command line takes one of the forms:

$$\left\{ \begin{array}{l} \text{<command>} \\ \text{<command> <filename>} \\ \text{<command> <filename> .<filetype>} \end{array} \right\}$$

Figure 1. CP/M Memory Organization



entry: the principal entry point to FDOS is at location 0005 which contains a JMP to fbase. The address field at location 0006 can be used to determine the size of available memory, assuming the CCP is being overlayed.

Note: The exact addresses for boot, tbase, cbase, fbase, and entry vary with the CP/M version (see Section 6. for version correspondence).

Where <command> is either a built-in command (e.g., DIR or TYPE), or the name of a transient command or program. If the <command> is a built-in function of CP/M, it is executed immediately; otherwise the CCP searches the currently addressed disk for a file by the name

<command>.COM

If the file is found, it is assumed to be a memory image of a program which executes in the TPA, and thus implicitly originates at tbase in memory (see the CP/M LOAD command). The CCP loads the COM file from the diskette into memory starting at tbase, and extending up to address cbase.

If the <command> is followed by either a <filename> or <filename>.<filetype>, then the CCP prepares a file control-block (FCB) in the system information area of memory. This FCB is in the form required to access the file through the FDOS, and is given in detail in Section 3.2.

The program then executes, perhaps using the I/O facilities of the FDOS. If the program uses no FDOS facilities, then the entire remaining memory area is available for data used by the program. If the FDOS is to remain in memory, then the transient program can use only up to location fbase as data.* In any case, if the CCP area is used by the transient, the entire CP/M system must be reloaded upon the transient's completion. This system reload is accomplished by a direct branch to location "boot" in memory.

The transient uses the CP/M I/O facilities to communicate with the operator's console and peripheral devices, including the floppy disk subsystem. The I/O system is accessed by passing a "function number" and an "information address" to CP/M through the address marked "entry" in Figure 1. In the case of a disk read, for example, the transient program sends the number corresponding to a disk read, along with the address of an FCB, and CP/M performs the operation, returning with either a disk read complete indication or an error number indicating that the disk operation was unsuccessful. The function numbers and error indicators are given in detail in Section 3.3.

1.3 Operating System Facilities

CP/M facilities which are available to transients are divided into two categories: BIOS operations, and BDOS primitives. The BIOS operations are listed first:**

* Address "entry" contains a jump to the lowest address in the FDOS, and thus "entry+1" contains the first FDOS address which cannot be overlaid.

**The device support (exclusive of the disk subsystem) corresponds exactly to Intel's peripheral definition, including I/O port assignment and status byte format (see the Intel manual which discusses the Intellec MDS hardware environment).

```

Read Console Character
Write Console Character
Read Reader Character
Write Punch Character
Write List Device Character
Set I/O Status
Interrogate Device Status
Print Console Buffer
Read Console Buffer
Interrogate Console Status

```

The exact details of BIOS access are given in Section 2. The BDOS primitives include the following operations:

```

Disk System Reset
Drive Select
File Creation
File Open
File Close
Directory Search
File Delete
File Rename
Read Record
Write Record
Interrogate Available Disks
Interrogate Selected Disk
Set DMA Address

```

The details of BDOS access are given in Section 3.

2. BASIC I/O FACILITIES

Access to common peripherals is accomplished by passing a function number and information address to the BIOS. In general, the function number is passed in Register C, while the information address is passed in Register pair D,E. Note that this conforms to the PL/M Conventions for parameter passing, and thus the following PL/M procedure is sufficient to link to the BIOS when a value is returned:

```

DECLARE ENTRY LITERALLY '0005H'; /* MONITOR ENTRY */

MON2: PROCEDURE (FUNC, INFO) BYTE;
      DECLARE FUNC BYTE, INFO ADDRESS;
      GO TO ENTRY;

      END MON2;

```

or

```

MON1:  PROCEDURE (FUNC,INFO);
        DECLARE FUNC BYTE, INFO ADDRESS;
        GO TO ENTRY;
        END MON1

```

if no returned value is expected.

2.1 Direct and Buffered I/O.

The BIOS entry points are given in Table I. In the case of simple character I/O to the console, the BIOS reads the console device, and removes the parity bit. The character is echoed back to the console, and tab characters (control-I) are expanded to tab positions starting at column one and separated by eight character positions. The I/O status byte takes the form shown in Table I, and can be programmatically interrogated or changed. The buffered read operation takes advantage of the CP/M line editing facilities. That is, the program sends the address of a read buffer whose first byte is the length of the buffer. The second byte is initially empty, but is filled-in by CP/M to the number of characters read from the console after the operation (not including the terminating carriage-return). The remaining positions are used to hold the characters read from the console. The BIOS line editing functions which are performed during this operation are given below:

```

break      - line delete and transmit
rubout     - delete last character typed, and echo
control-C  - system reboot
control-U  - delete entire line
control-E  - return carriage, but do not transmit
             buffer (physical carriage return)
<cr>      - transmit buffer

```

The read routine also detects control character sequences other than those shown above, and echos them with a preceding "!" symbol. The print entry point allows an entire string of symbols to be printed before returning from the BIOS. The string is terminated by a "\$" symbol.

2.2 A Simple Example

As an example, consider the following PL/M procedures and procedure calls which print a heading, and successively read the console buffer. Each console buffer is then echoed back in reverse order:

```

PRINTCHAR:  PROCEDURE (B);
            /* SEND THE ASCII CHARACTER B TO THE CONSOLE */
            DECLARE B BYTE;
            CALL MON1(2,B);
            END PRINTCHAR;

CRLF:  PROCEDURE;
        /* SEND CARRIAGE-RETURN-LINE-FEED CHARACTERS */
        CALL PRINTCHAR (0DH); CALL PRINTCHAR (0AH);
        END CRLF;

PRINT:  PROCEDURE (A);
        /* PRINT THE BUFFER STARTING AT ADDRESS A */
        DECLARE A ADDRESS;
        CALL MON1(9,A);
        END PRINT;

DECLARE  RDBUFF (130) BYTE;

READ:  PROCEDURE;
        /* READ CONSOLE CHARACTERS INTO 'RDBUFF' */
        RDBUFF=128; /* FIRST BYTE SET TO BUFFER LENGTH */
        CALL MON1(10,.RDBUFF);
        END READ;

DECLARE I BYTE;
CALL CRLF;
        CALL PRINT (.'TYPE INPUT LINES $');
        DO WHILE 1; /* INFINITE LOOP-UNTIL CONTROL-C */
        CALL CRLF; CALL PRINTCHAR ('*'); /* PROMPT WITH '*' */
        CALL READ; I = RDBUFF(1);
            DO WHILE (I := I -1) <> 255;
            CALL PRINTCHAR (RDBUFF(I+2));
            END;
        END;

```

The execution of this program might proceed as follows:

```

TYPE INPUT LINES
*HELLO,
OLLEH
*WALL WALLA WASH,
HSAW ALLAW ALLAW
*MOM WOW,
WOW MOM
*↑C                (system reboot)

```

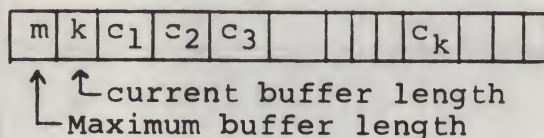
TABLE I
BASIC I/O OPERATIONS

FUNCTION/ NUMBER	ENTRY PARAMETERS	RETURNED VALUE	TYPICAL CALL
Read Console 1	None	ASCII Character	I = MON2(1,0)
Write Console 2	ASCII Character	None	CALL MON1(2,'A')
Read Reader 3	None	ASCII Character	I = MON2(3,0)
Write Punch 4	ASCII Character	None	CALL MON1(4,'B')
Write List 5	ASCII Character	None	CALL MON1(5,'C')
Get I/O Status 7	None	I/O Status Byte	IOSTAT=MON2(7,0)
Set I/O Status 8	I/O Status Byte	None	CALL MON1(8,IOSTAT)
Print Buffer 9	Address of string termi- nated by '\$'	None	CALL MON1(9, .'PRINT THIS \$')

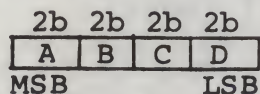
TABLE I (continued)

FUNCTION/ NUMBER	ENTRY PARAMETERS	RETURNED VALUE	TYPICAL CALL
Read Buffer 10	Address of Read Buffer* (See Note ₁)	Read buffer is filled to maxi- mum length with console charac- ters	CALL MON1(10, .RDBUFF);
Interrogate Console Ready 11	None	Byte value with least signifi- cant bit = 1 (true) if con- sole character is ready	I = MON2(11,0)

Note₁: Read buffer is a sequence of memory locations of the form:



Note₂: The I/O status byte is defined as three fields A,B,C, and D



requiring two bits each, listed from most significant to least significant bit, which define the current device assignment as follows:

$$\begin{array}{l}
 \text{D} \\
 \text{Console}
 \end{array}
 = \left\{ \begin{array}{ll} 0 & \text{TTY} \\ 1 & \text{CRT} \\ 2 & \text{BATCH} \\ 3 & - \end{array} \right\}
 \quad
 \begin{array}{l}
 \text{C} \\
 \text{Reader}
 \end{array}
 = \left\{ \begin{array}{ll} 0 & \text{TTY} \\ 1 & \text{FAST READER} \\ 2 & - \\ 3 & - \end{array} \right\}
 \quad
 \begin{array}{l}
 \text{B} \\
 \text{Punch}
 \end{array}
 = \left\{ \begin{array}{ll} 0 & \text{TTY} \\ 1 & \text{FAST PUNCH} \\ 2 & - \\ 3 & - \end{array} \right\}
 \quad
 \begin{array}{l}
 \text{A} \\
 \text{List}
 \end{array}
 = \left\{ \begin{array}{ll} 0 & \text{TTY} \\ 1 & \text{CRT} \\ 2 & - \\ 3 & - \end{array} \right\}$$

3. DISK I/O FACILITIES

The BDOS section of CP/M provides access to files stored on diskettes. The discussion which follows gives the overall file organization, along with file access mechanisms.

3.1 File Organization

CP/M implements a named file structure on each diskette, providing a logical organization which allows any particular file to contain any number of records, from completely empty, to the full capacity of a diskette. Each diskette is logically distinct, with a complete operating system, disk directory, and file data area. The disk file names are in two parts: the <filename> which can be from one to eight alphanumeric characters, and the <filetype> which consists of zero through three alphanumeric characters. The <filetype> names the generic category of a particular file, while the <filename> distinguishes a particular file within the category. The <filetype>s listed below give some generic categories which have been established, although they are generally arbitrary:

ASM	assembler source file
PRN	assembler listing file
HEX	assembler or PL/M machine code in "hex" format
BAS	BASIC Source file
INT	BASIC Intermediate file
COM	Memory image file (i.e., "Command" file for transients, produced by LOAD)
BAK	Backup file produced by editor (see ED manual)
\$\$\$	Temporary files created and normally erased by editor and utilities

Thus, the name

X.ASM

is interpreted as an assembly language source file by the CCP with <filename> X.

The files in CP/M are organized as a logically contiguous sequence of 128 byte records (although the records may not be physically contiguous on the diskette), which are normally read or written in sequential order. Random access is allowed under CP/M however, as described in Section 3.4. No particular format within records is assumed by CP/M, although some transients expect particular formats:

- (1) Source files are considered a sequence of ASCII characters, where each "line" of the source file is followed by carriage-return-line-feed characters. Thus, one 128 byte CP/M record could contain several logical lines of source text. Machine code "hex" tapes are also assumed to be in this format, although the loader does not require the carriage-return-line-feed characters. End of text is given by the character control-z, or real end-of-file returned by CP/M.

and

- (2) COM files are assumed to be absolute machine code in memory image form, starting at tbase in memory. In this case, control-z is not considered an end of file, but instead is determined by the actual space allocated to the file being accessed.

3.2 File Control Block Format

Each file being accessed through CP/M has a corresponding file control block (FCB) which provides name and allocation information for all file operations. The FCB is a 33-byte area in the transient program's memory space which is set up for each file. The FCB format is given in Figure 2. When accessing CP/M files, it is the programmer's responsibility to fill the lower 16 bytes of the FCB, along with the CR field. Normally, the FN and FT fields are set to the ASCII <filename> and <filetype>, while all other fields are set to zero. Each FCB describes up to 16K bytes of a particular file (0 to 128 records of 128 bytes each), and, using automatic mechanisms of CP/M, up to 15 additional extensions of the file can be addressed. Thus, each FCB can potentially describe files up to 256K bytes (which is slightly larger than the diskette capacity).

FCB's are stored in a directory area of the diskette, and are brought into central memory before file operations (see the OPEN and MAKE commands) then updated in memory as file operations proceed, and finally recorded on the diskette at the termination of the file operation (see the CLOSE command). This organization makes CP/M file organization highly reliable, since diskette file integrity can only be disrupted in the unlikely case of hardware failure during update of a single directory entry.

It should be noted that the CCP constructs an FCB for all transients by scanning the remainder of the line following the transient name for a <filename> or <filename>.<filetype> combination. Any field not specified is assumed to be all blanks. A properly formed FCB is set up at location tfcb (see Section 6), with an assumed I/O buffer at tbuff. The transient can use tfcb as an address in subsequent input or output operations on this file.

In addition to the default fcb which is set-up at address tfcb, the CCP also constructs a second default fcb at address tfcb+16 (i.e., the disk map field of the fcb at tbase). Thus, if the user types

PROGNAME X.ZOT Y.ZAP

the file PROGNAME.COM is loaded to the TPA, and the default fcb at tfcb is initialized to the filename X with filetype ZOT. Since the user typed a second file name, the 16 byte area beginning at tfcb + 16₁₀ is also initialized with the filename Y and filetype ZAP. It is the responsibility of the program to move this second filename and filetype to another area (usually a separate file control block) before opening the file which begins at tbase, since the open operation will fill the disk map portion, thus overwriting the second name and type.

If no file names were specified in the original command, then the fields beginning at tfcb and tfcb + 16 both contain blanks (20H). If one file name was specified, then the field at tfcb + 16 contains blanks. If the filetype is omitted, then the field is assumed to contain blanks. In all cases, the CCP translates lower case alphabets to upper case to be consistent with the CP/M file naming conventions.

As an added programming convenience, the default buffer at tbuff is initialized to hold the entire command line past the program name. Address tbuff contains the number of characters, and tbuff+1, tbuff+2, ..., contain the remaining characters up to, but not including, the carriage return. Given that the above command has been typed at the console, the area beginning at tbuff is set up as follows:

tbuff:

+0	+1	+2	+3	+4	+5	+6	+7	+8	+9	+10	+11	+12	+13	+14	+15
12	Ø	X	.	Z	O	T	Ø	Y	.	Z	A	P	?	?	?

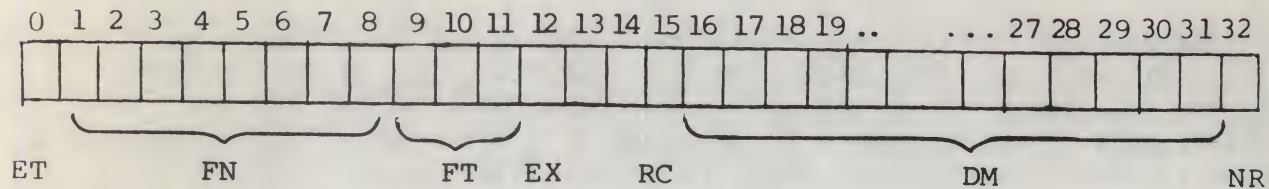
where 12 is the number of valid characters (in binary), and Ø represents an ASCII blank. Characters are given in ASCII upper case, with uninitialized memory following the last valid character.

Again, it is the responsibility of the program to extract the information from this buffer before any file operations are performed since the FDOS uses the tbuff area to perform directory functions.

In a standard CP/M system, the following values are assumed:

boot:	0000H	bootstrap load (warm start)
entry:	0005H	entry point to FDOS
tfcb:	005CH	first default file control block
tfcb+16	006CH	second file name
tbuff	0080H	default buffer address
tbase:	0100H	base of transient area

Figure 2. File Control Block Format



<u>FIELD</u>	<u>FCB POSITIONS</u>	<u>PURPOSE</u>
ET	0	Entry type (currently not used, but assumed zero)
FN	1-8	File name, padded with ASCII blanks
FT	9-11	File type, padded with ASCII blanks
EX	12	File extent, normally set to zero
	13-14	Not used, but assumed zero
RC	15	Record count is current extent Size (0 to 128 records)
DM	16-31	Disk allocation map, filled-in and used by CP/M
NR	32	Next record number to read or write

3.3 Disk Access Primitives

Given that a program has properly initialized the FCB's for each of its files, there are several operations which can be performed, as shown in Table II. In each case, the operation is applied to the currently selected disk (see the disk select operation in Table II), using the file information in a specific FCB. The following PL/M program segment, for example, copies the contents of the file X.Y to the (new) file NEW.FIL:

```
DECLARE RET BYTE;
```

```
OPEN:      PROCEDURE (A)
            DECLARE A ADDRESS;
            RET=MON2(15,A);
            END OPEN;
```

```
CLOSE:     PROCEDURE (A);
            DECLARE A ADDRESS;
            RET=MON2(16,A);
            END;
```

```
MAKE:      PROCEDURE (A);
            DECLARE A ADDRESS;
            RET=MON2(22,A);
            END MAKE;
```

```
DELETE:    PROCEDURE (A);
            DECLARE A ADDRESS;
            /* IGNORE RETURNED VALUE */
            CALL MON1(19,A);
            END DELETE;
```

```
READBF:    PROCEDURE (A);
            DECLARE A ADDRESS;
            RET=MON2(20,A);
            END READBF;
```

```
WRITEBF:   PROCEDURE (A);
            DECLARE A ADDRESS;
            RET=MON2(21,A);
            END WRITEBF;
```

```
INIT:      PROCEDURE;
            CALL MON1(13,0);
            END INIT;
```

```
/* SET UP FILE CONTROL BLOCKS */
DECLARE FCB1 (33) BYTE
    INITIAL (0,'X      ','Y  ',0,0,0,0),
    FCB2 (33) BYTE
    INITIAL (0,'NEW    ','FIL',0,0,0,0);
```

```

CALL INIT;
/* ERASE 'NEW.FIL' IF IT EXISTS */
CALL DELETE (.FCB2);
/* CREATE 'NEW.FIL' AND CHECK SUCCESS */
CALL MAKE (.FCB2);
IF RET = 255 THEN CALL PRINT (.'NO DIRECTORY SPACE $');
ELSE
DO; /* FILE SUCCESSFULLY CREATED, NOW OPEN 'X.Y' */
CALL OPEN (.FCB1);
IF RET = 255 THEN CALL PRINT (.'FILE NOT PRESENT $');
ELSE
DO; /* FILE X.Y FOUND AND OPENED, SET
NEXT RECORD TO ZERO FOR BOTH FILES */
FCB1(32), FCB2(32) = 0;
/* READ FILE X.Y UNTIL EOF OR ERROR */
CALL READBF (.FCB1); /*READ TO 80H*/
DO WHILE RET = 0;
CALL WRITEBF (.FCB2) /*WRITE FROM 80H*/
IF RET = 0 THEN /*GET ANOTHER RECORD*/
CALL READBF (.FCB1); ELSE
CALL PRINT (.'DISK WRITE ERROR $');
END;
IF RET < >1 THEN CALL PRINT (.'TRANSFER ERROR $');
ELSE
DO; CALL CLOSE (.FCB2);
IF RET = 255 THEN CALL PRINT (.'CLOSE ERROR$');
END;
END;
END;
EOF

```

This program consists of a number of utility procedures for opening, closing, creating, and deleting files, as well as two procedures for reading and writing data. These utility procedures are followed by two FCB's for the input and output files. In both cases, the first 16 bytes are initialized to the <filename> and <filetype> of the input and output files. The main program first initializes the disk system, then deletes any existing copy of "NEW.FIL" before starting. The next step is to create a new directory entry (and empty file) for "NEW.FIL". If file creation is successful, the input file "X.Y" is opened. If this second operation is also successful, then the disk to disk copy can proceed. The NR fields are set to zero so that the first record of each file is accessed on subsequent disk I/O operations. The first call to READBF fills the (implied) DMA buffer at 80H with the first record from X.Y. The loop which follows copies the record at 80H to "NEW.FIL" and then reports any errors, or reads another 128 bytes from X.Y. This transfer operation continues until either all data has been transferred, or an error condition arises. If an error occurs, it is reported; otherwise the new file is closed and the program halts.

TABLE II

DISK ACCESS PRIMITIVES

FUNCTION/NUMBER	ENTRY PARAMETERS	RETURNED VALUE	TYPICAL CALL
Lift Head 12	None	None Head is lifted from current drive	CALL MON2(12,0)
Initialize BDOS and select disk "A" Set DMA address to 80H 13	None	None Side effect is that disk A is "logged- in" while all others are considered "off- line"	CALL MON1(13,0)
Log-in and select disk X 14	An integer value cor- responding to the disk to log-in: A=0, B=1, C=2, etc.	None Disk X is considered "on-line" and selec- ted for subsequent file operations	CALL MON1(14,1) (log-in disk "B")
Open file 15	Address of the FCB for the file to be accessed	Byte address of the FCB in the directory, if found, or 255 if file not present. The DM bytes are set by the BDOS.	I = MON2(15,.FCB)
Close file 16	Address of an FCB which has been pre- viously created or opened	Byte address of the directory entry cor- responding to the FCB, or 255 if not present	I = MON2(16,.FCB)

TABLE II (continued)

FUNCTION/NUMBER	ENTRY PARAMETERS	RETURNED VALUE	TYPICAL CALL
Search for file 17	Address of FCB containing <filename> and <filetype> to match. ASCII "?" in FCB matches any character.	Byte address of first FCB in directory that matches input FCB, if any; otherwise 255 indicates no match.	I = MON2(17,.FCB)
Search for next occurrence 18	Same as above, but called after function 17 (no other intermediate BDOS calls allowed)	Byte address of next	I = MON2(18,.FCB)
Delete File 19	Address of FCB containing <filename> and <filetype> of file to delete from diskette	None	I = MON2(19,.FCB)
Read Next Record 20	Address of FCB of a successfully OPENED file, with NR set to the next record to read (see note ₁)	0 = successful read 1 = read past end of file 2 = reading unwritten data in random access	I = MON2(20,.FCB)

Note₁: The I/O operations transfer data to/from address 80H for the next 128 bytes unless the DMA address has been altered (see function 26). Further, the NR field of the FCB is automatically incremented after the operation. If the NR field exceeds 128, the next extent is opened automatically and the NR field is reset to zero.

TABLE II (continued)

FUNCTION/NUMBER	ENTRY PARAMETERS	RETURNED VALUE	TYPICAL CALL
Write Next Record 21	Same as above, except NR is set to the next record to write	0 = successful write 1 = error in extending file 2 = end of disk data 255 = no more directory space	I = MON2(21, .FCB)
Make File 22	Address of FCB with <filename> and <file-type> set. Directory entry is created, the file is initialized to empty.	Byte address of directory entry allocated to the FCB, or 255 if no directory space is available	I = MON2(22, .FCB)
Rename FCB 23	Address of FCB with old FN and FT in first 16 bytes, and new FN and FT in second 16 bytes	None	I = MON2(23, .FCB)

TABLE II (continued)

FUNCTION/NUMBER	ENTRY PARAMETERS	RETURNED VALUE	TYPICAL CALL
Interrogate log- in vector 24	None	Byte value with "1" in bit positions of "on line" disks, with least signi- ficant bit corres- ponding to disk "A"	I = MON2(24,0)
Set DMA address 26	Address of 128 byte DMA buffer	None Subsequent disk I/O takes place at spe- cified address in memory	CALL MON1(26,2000H)
Interrogate Allocation 27	None	Address of the allo- cation vector for the current disk (used by STATUS com- mand)	MON3: PROCEDURE(...) ADDRESS; A = MON3(27,0);
Interrogate Drive number 25	None	Disk number of currently logged disk (i.e., the drive which will be used for the next disk operation	I = MON2(25,0);

3.4 Random Access

Recall that a single FCB describes up to a 16K segment of a (possibly) larger file. Random access within the first 16K segment is accomplished by setting the NR field to the record number of the record to be accessed before the disk I/O takes place. Note, however, that if the 128th record is written, then the BDOS automatically increments the extent field (EX), and opens the next extent, if possible. In this case, the program must explicitly decrement the EX field and re-open the previous extent. If random access outside the first 16K segment is necessary, then the extent number e be explicitly computed, given an absolute record number r as

$$e = \left\lfloor \frac{r}{128} \right\rfloor$$

or equivalently,

$$e = \text{SHR}(r, 7)$$

this extent number is then placed in the EX field before the segment is opened. The NR value n is then computed as

$$n = r \bmod 128$$

or

$$n = r \text{ AND } 7\text{FH}.$$

When the programmer expects considerable cross-segment accesses, it may save time to create an FCB for each of the 16K segments, open all segments for access, and compute the relevant FCB from the absolute record number r .

4. SYSTEM GENERATION

As mentioned previously, every diskette used under CP/M is assumed to contain the entire system (excluding transient commands) on the first two tracks. The operating system need not be present, however, if the diskette is only used as secondary disk storage on drives B, C, ..., since the CP/M system is loaded only from drive A.

The CP/M file system is organized so that an IBM-compatible diskette from the factory (or from a vendor which claims IBM compatibility) looks like a diskette with an empty directory. Thus, the user must first copy a version of the CP/M system from an existing diskette to the first two tracks of the new diskette, followed by a sequence of copy operations, using PIP, which transfer the transient command files from the original diskette to the new diskette.

NOTE: before you begin the CP/M copy operation, read your Licensing Agreement. It gives your exact legal obligations when making reproductions of CP/M in whole or in part, and specifically requires that you place the copyright notice

Copyright (c), 1976
Digital Research

on each diskette which results from the copy operation.

4.1. Initializing CP/M from an Existing Diskette

The first two tracks are placed on a new diskette by running the transient command SYSGEN, as described in the document "An Introduction to CP/M Features and Facilities." The SYSGEN operation brings the CP/M system from an initialized diskette into memory, and then takes the memory image and places it on the new diskette.

Upon completion of the SYSGEN operation, place the original diskette on drive A, and the initialized diskette on drive B. Reboot the system; the response should be

A>

indicating that drive A is active. Log into drive B by typing

B:

and CP/M should respond with

B>

indicating that drive B is active. If the diskette in drive B is factory fresh, it will contain an empty directory. Non-standard diskettes may, however, appear as full directories to CP/M, which can be emptied by typing

ERA *.*

when the diskette to be initialized is active. Do not give the ERA command if you wish to preserve files on the new diskette since all files will be erased with this command.

After examining disk B, reboot the CP/M system and return to drive A for further operations.

The transient commands are then copied from drive A to drive B using the PIP program. The sequence of commands shown below, for example, copy the principal programs from a standard CP/M diskette to the new diskette:

```
A>PIP,
*B:STAT.COM=STAT.COM,
*B:PIP.COM=PIP.COM,
*B:LOAD.COM=LOAD.COM,
*B:ED.COM=ED.COM,
```

```

*B:ASM.COM=ASM.COM,
*B:SYSGEN.COM=SYSGEN.COM,
*B:DDT.COM=DDT.COM,
*,
A>

```

The user should then log in disk B, and type the command

```
DIR *.*,
```

to ensure that the files were transferred to drive B from drive A. The various programs can then be tested on drive B to check that they were transferred properly.

Note that the copy operation can be simplified somewhat by creating a "submit" file which contains the copy commands. The file could be named GEN.SUB, for example, and might contain

```

SYSGEN,
PIP B:STAT.COM=STAT.COM,
PIP B:PIP.COM=PIP.COM,
PIP B:LOAD.COM=LOAD.COM,
PIP B:ED.COM=ED.COM,
PIP B:ASM.COM=ASM.COM,
PIP B:SYSGEN.COM=SYSGEN.COM,
PIP B:DDT.COM=DDT.COM,

```

The generation of a new diskette from the standard diskette is then done by typing simply

```
SUBMIT GEN,
```

5. CP/M ENTRY POINT SUMMARY

The functions shown below summarize the functions of the FDOS. The function number is passed in Register C (first parameter in PL/M), and the information is passed in Registers D,E (second PL/M parameter). Single byte results are returned in Register A. If a double byte result is returned, then the high-order byte comes back in Register B (normal PL/M return). The transient program enters the FDOS through location "entry" (see Section 7.) as shown in Section 2. for PL/M, or

CALL entry

in assembly language. All registers are altered in the FDOS.

<u>Function</u>	<u>Number</u>	<u>Information</u>	<u>Result</u>
0	System Reset		
1	Read Console		ASCII character
2	Write Console	ASCII character	
3	Read Reader		ASCII character
4	Write Punch	ASCII character	
5	Write List	ASCII character	
6	(not used)		
7	Interrogate I/O Status		I/O Status Byte
8	Alter I/O Status	I/O Status Byte	
9	Print Console Buffer	Buffer Address	
10	Read Console Buffer	Buffer Address	
11	Check Console Status		True if character Ready
12	Lift Disk Head		
13	Reset Disk System		
14	Select Disk	Disk number	
15	Open File	FCB Address	Completion Code
16	Close File	" "	" "
17	Search First	" "	" "
18	Search Next	" "	" "
19	Delete File	" "	" "
20	Read Record	" "	" "
21	Write Record	" "	" "
22	Create File	" "	" "
23	Rename File	" "	" "
24	Interrogate Login		Login Vector
25	Interrogate Disk		Selected Disk Number
26	Set DMA Address	DMA Address	
27	Interrogate Allocation		Address of Allocation Vector

6. ADDRESS ASSIGNMENTS

The standard distribution version of CP/M is organized for an Intel MDS microcomputer developmental system with 16K of main memory, and two diskette drives. Larger systems are available in 16K increments, providing management of 32K, 48K, and 64K systems (the largest MDS system is 62K since the ROM monitor provided with the MDS resides in the top 2K of the memory space). For each additional 16K increment, add 4000H to the values of cbase and fbase.

The address assignments are

boot = 0000H	warm start operation
tfcb = 005CH	default file control block location
tbuf= 0080H	default buffer location
tbase= 0100H	base of transient program area
cbase= 2900H	base of console command processor
fbase= 3200H	base of disk operating system
entry= 0005H	entry point to disk system from user programs

7. SAMPLE PROGRAMS

This section contains two sample programs which interface with the CP/M operating system. The first program is written in assembly language, and is the source program for the DUMP utility. The second program is the CP/M LOAD utility, written in PL/M.

The assembly language program begins with a number of "equates" for system entry points and program constants. The equate

```
BDOS    EQU    0005H
```

for example, gives the CP/M entry point for peripheral I/O functions. The default file control block address is also defined (FCB), along with the default buffer address (BUFF). Note that the program is set up to run at location 100H, which is the base of the transient program area. The stack is first set-up by saving the entry stack pointer into OLDSP, and resetting SP to the local stack. The stack pointer upon entry belongs to the console command processor, and need not be saved unless control is to return to the CCP upon exit. That is, if the program terminates with a reboot (branch to location 0000H) then the entry stack pointer need not be saved.

The program then jumps to MAIN, past a number of subroutines which are listed below:

- BREAK - when called, checks to see if there is a console character ready. BREAK is used to stop the listing at the console
- PCHAR - print the character which is in register A at the console.
- CRLF - send carriage return and line feed to the console
- PNIB - print the hexadecimal value in register A in ASCII at the console
- PHEX - print the byte value (two ASCII characters) in register A at the console
- ERR - print error flag #n at the console, where n is
 - 1 if file cannot be opened
 - 2 if disk read error occurred
- GNB - get next byte of data from the input file. If the IBP (input buffer pointer) exceeds the size of the input buffer, then another disk record of 128 bytes is read. Otherwise, the next character in the buffer is returned. IBP is updated to point to the next character.

The MAIN program then appears, which begins by calling SETUP. The SETUP subroutine, discussed below, opens the input file and checks for errors. If the file is opened properly, the GLOOP (get loop) label gets control.

On each successive pass through the GLOOP label, the next data byte is fetched using GNB and save in register B. The line addresses are listed every sixteen bytes, so there must be a check to see if the least significant 4 bits is zero on each output. If so, the line address is taken from registers h and l, and typed at the left of the line. In all cases, the byte which was previously saved in register B is brought back to register A, following label NONUM, and printed in the output line. The cycle through GLOOP continues until an end of file condition is detected in DISKR, as described below. Thus, the output lines appear as

```
0000  bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb
0010  bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb
```

...

until the end of file.

The label FINIS gets control upon end of file. CRLF is called first to return the carriage from the last line output. The CCP stack pointer is then reclaimed from OLDSP, followed by a RET to return to the console command processor. Note that a JMP 0000H could be used following the FINIS label, which would cause the CP/M system to be brought in again from the diskette (this operation is necessary only if the CCP has been overlaid by data areas).

The file control block format is then listed (FCBDN ... FCBLN) which overlays the fcb at location 05CH which is setup by the CCP when the DUMP program is initiated. That is, if the user types

DUMP X.Y

then the CCP sets up a properly formed fcb at location 05CH for the DUMP (or any other) program when it goes into execution. Thus, the SETUP subroutine simply addresses this default fcb, and calls the disk system to open it. The DISKR (disk read) routine is called whenever GNB needs another buffer full of data. The default buffer at location 80H is used, along with a pointer (IBP) which counts bytes as they are processed. Normally, an end of file condition is taken as either an ASCII 1AH (control-z), or an end of file detection by the DOS. The file dump program, however, stops only on a DOS end of file.

```

; FILE DUMP PROGRAM, READS AN INPUT FILE AND PRINTS IN
;
; COPYRIGHT (C), DIGITAL RESEARCH, 1975, 1976
;

```

25

```

0100      ORG      100H
0005 =    BDOS    EQU      0005H      ;DOS ENTRY POINT
000F =    OPENF   EQU      15         ;FILE OPEN
0014 =    READF   EQU      20         ;READ FUNCTION
0002 =    TYPEF   EQU      2         ;TYPE FUNCTION
0001 =    CONS    EQU      1         ;READ CONSOLE
000B =    BRKF    EQU      11        ;BREAK KEY FUNCTION (TRUE IF CHAR READY)
;

```

```

005C =    FCB     EQU      5CH        ;FILE CONTROL BLOCK ADDRESS
0080 =    BUFF    EQU      80H        ;INPUT DISK BUFFER ADDRESS
;

```

```

; SET UP STACK
0100 210000 LXI      H,0
0103 39     DAD     SP
0104 220F01 SHLD    OLDSP
0107 315101 LXI      SP,STKTOP
010A C3C401 JMP     MAIN
;

```

```

; VARIABLES
010D      IBP:    DS      2           ;INPUT BUFFER POINTER
;

```

```

; STACK AREA
010F      OLDSP:  DS      2
0111      STACK: DS      64
0151 =    STKTOP EQU      $
;

```

```

; SUBROUTINES
;
BREAK: ;CHECK BREAK KEY (ACTUALLY ANY KEY WILL DO)
0151 E5D5C5 PUSH H! PUSH D! PUSH B; ENVIRONMENT SAVED
0154 0E0B   MVI     C,BRKF
0156 CD0500 CALL    BDOS
0159 C1D1E1 POP B! POP D! POP H; ENVIRONMENT RESTORED
015C C9     RET
;

```

```

; PCHAR: ;PRINT A CHARACTER
015D E5D5C5 PUSH H! PUSH D! PUSH B; SAVED
0160 0E02   MVI     C,TYPEF
0162 5F     MOV     E,A
0163 CD0500 CALL    BDOS
0166 C1D1E1 POP B! POP D! POP H; RESTORED
0169 C9     RET
;

```

```

; CRLF:
016A 3E0D   MVI     A,0DH
016C CD5D01 CALL    PCHAR
016F 3E0A   MVI     A,0AH
0171 CD5D01 CALL    PCHAR
0174 C9     RET
;

```

```

;
; PNIB: ;PRINT NIBBLE IN REG A
0175 E60F   ANI     0FH      ;LOW 4 BITS
0177 FE0A   CPI     10
0179 D28101 JNC     P10

```

```

; LESS THAN OR EQUAL TO 9
017C C630      ADI      '0'
017E C38301    JMP      PRN

;
; GREATER OR EQUAL TO 10
0181 C637      P10:    ADI      'A' - 10
0183 CD5D01    PRN:    CALL     PCHAR
0186 C9        RET

;
PHEX:          ;PRINT HEX CHAR IN REG A
0187 F5        PUSH     PSW
0188 0F        RRC
0189 0F        RRC
018A 0F        RRC
018B 0F        RRC
018C CD7501    CALL     PNIB      ;PRINT NIBBLE
018F F1        POP      PSW
0190 CD7501    CALL     PNIB
0193 C9        RET

;
ERR:           ;PRINT ERROR MESSAGE
0194 CD6A01    CALL     CRLF
0197 3E23      MVI      A, '#'
0199 CD5D01    CALL     PCHAR
019C 78        MOV      A,B
019D C630      ADI      '0'
019F CD5D01    CALL     PCHAR
01A2 CD6A01    CALL     CRLF
01A5 C3F701    JMP      FINIS

;
GNB:           ;GET NEXT BYTE
01A8 3A0D01    LDA      IBP
01AB FE80      CPI      80H
01AD C2B401    JNZ      G0
; READ ANOTHER BUFFER
;
;
01B0 CD1602    CALL     DISKR
01B3 AF        XRA      A

;
G0:           ;READ THE BYTE AT BUFF+REG A
01B4 5F        MOV      E,A
01B5 1600      MVI      D,0
01B7 3C        INR      A
01B8 320D01    STA      IBP
; POINTER IS INCREMENTED
; SAVE THE CURRENT FILE ADDRESS
01BB E5        PUSH     H
01BC 218000    LXI      H,BUFF
01BF 19        DAD      D
01C0 7E        MOV      A,M
; BYTE IS IN THE ACCUMULATOR
;
; RESTORE FILE ADDRESS AND INCREMENT
01C1 E1        POP      H
01C2 23        INX      H
01C3 C9        RET

;
MAIN:          ; READ AND PRINT SUCCESSIVE BUFFERS
01C4 CDFF01    CALL     SETUP      ;SET UP INPUT FILE

```

```

01C7 3E80          MVI      A,80H
01C9 320D01        STA      IBP      ;SET BUFFER POINTER TO 80H
01CC 21FFFF        LXI      H,0FFFFH ;SET TO -1 TO START
;
GLOOP:
01CF CDA801        CALL     GNB
01D2 47            MOV      B,A
;                PRINT HEX VALUES
;
01D3 7D            MOV      A,L
01D4 E60F          ANI      0FH      ;CHECK LOW 4 BITS
01D6 C2EB01        JNZ      NONUM
;                PRINT LINE NUMBER
01D9 CD6A01        CALL     CRLF
;
;                CHECK FOR BREAK KEY
01DC CD5101        CALL     BREAK
01DF 0F            RRC
01E0 DAF701        JC       FINIS    ;DON'T PRINT ANY MORE
;
01E3 7C            MOV      A,H
01E4 CD8701        CALL     PHEX
01E7 7D            MOV      A,L
01E8 CD8701        CALL     PHEX
NONUM:
01EB 3E20          MVI      A,
01ED CD5D01        CALL     PCHAR
01F0 78            MOV      A,B
01F1 CD8701        CALL     PHEX
01F4 C3CF01        JMP      GLOOP
;
EPSA: ;END PSA
;                END OF INPUT
FINIS:
01F7 CD6A01        CALL     CRLF
01FA 2A0F01        LHL      OLDSP
01FD F9            SPHL
01FE C9            RET
;
;
;                FILE CONTROL BLOCK DEFINITIONS
005C =             FCBDN EQU      FCB+0    ;DISK NAME
005D =             FCBFN EQU      FCB+1    ;FILE NAME
0065 =             FCBFT EQU      FCB+9    ;DISK FILE TYPE (3 CHARACTERS)
0068 =             FCBRL EQU      FCB+12   ;FILE'S CURRENT REEL NUMBER
006B =             FCBRC EQU      FCB+15   ;FILE'S RECORD COUNT (0 TO 128)
007C =             FCBCR EQU      FCB+32   ;CURRENT (NEXT) RECORD NUMBER (0 TO 127)
007D =             FCBLN EQU      FCB+33   ;FCB LENGTH
;
;
SETUP: ;SET UP FILE
;                OPEN THE FILE FOR INPUT
01FF 115C00        LXI      D,FCB
0202 0E0F          MVI      C,OPENF
0204 CD0500        CALL     BDOS
;                CHECK FOR ERRORS
0207 FEFF          CPI      255
0209 C21102        JNZ      OPNOK

```

```

020C 0601      ;      BAD OPEN
020E CD9401    MVI      B,1      ;OPEN ERROR
               CALL      ERR
               ;
OPNOK: ;OPEN IS OK.
0211 AF        XRA      A
0212 327C00    STA      FCBCR
0215 C9        RET
               ;
DISKR: ;READ DISK FILE RECORD
0216 E5D5C5    PUSH H! PUSH D! PUSH B
0219 115C00    LXI      D,FCB
021C 0E14      MVI      C,READF
021E CD0500    CALL     BDOS
0221 C1D1E1    POP B! POP D! POP H
0224 FE00      CPI      0      ;CHECK FOR ERRS
0226 C8        RZ
               ;
0227 FE01      MAY BE EOF
0229 CAF701    CPI      1
               JZ      FINIS
               ;
022C 0602      MVI      B,2      ;DISK READ ERROR
022E CD9401    CALL     ERR
               ;
0231          END

```

The PL/M program which follows implements the CP/M LOAD utility. The function is as follows. The user types

```
LOAD filename,
```

If filename.HEX exists on the diskette, then the LOAD utility reads the "hex" formatted machine code file and produces the file

```
filename.COM
```

where the COM file contains an absolute memory image of the machine code, ready for load and execution in the TPA. If the file does not appear on the diskette, the LOAD program types

```
SOURCE IS READER
```

and reads an Addmaster paper tape reader which contains the hex file.

The LOAD program is set up to load and run in the TPA, and, upon completion, return to the CCP without rebooting the system. Thus, the program is constructed as a single procedure called LOADCOM which takes the form

```
OFAH:
  LOADCOM: PROCEDURE;
    /* LIBRARY PROCEDURES */
    MON1: ...
    /* END LIBRARY PROCEDURES */
    MOVE: ...
    GETCHAR: ...
    PRINTNIB: ...
    PRINTHEX: ...
    PRINTADDR: ...
    RELOC: ...
    {
      SETMEM:
      READHEX:
      READBYTE:
      READCS:
      MAKEDOUBLE:
      DIAGNOSE:
    }
    END RELOC;

    DECLARE STACK(16) ADDRESS, SP ADDRESS;
    SP = STACKPTR; STACKPTR = .STACK(LENGTH(STACK));

    ...
    CALL RELOC;
    ...
    STACKPTR = SP;
    RETURN 0;
  END LOADCOM;
;
EOF
```

The label OFAH at the beginning sets the origin of the compilation to OFAH, which causes the first 6 bytes of the compilation to be ignored when loaded (i.e., the TPA starts at location 100H and thus OFAH,...,OFFH are deleted from the COM file). In a PL/M compilation, these 6 bytes are used to set up the stack pointer and branch around the subroutines in the program. In this case, there is only one subroutine, called LOADCOM, which results in the following machine memory image for LOAD

```

OFAH: LXI SP,plmstack      ;SET SP TO DEFAULT STACK
OFDH: JMP pastsubr         ;JUMP AROUND LOADCOM
100H: beginning of LOADCOM procedure
      ....
      end of LOADCOM procedure
      RET

pastsubr:
      EI
      HLT

```

Since the machine code between OFAH and OFFH is deleted in the load, execution actually begins at the top of LOADCOM. Note, however, that the initialization of the SP to the default stack has also been deleted; thus, there is a declaration and initialization of an explicit stack and stack pointer before the call to RELOC at the end of LOADCOM. This is necessary only if we wish to return to the CCP without a reboot operation; otherwise the origin of the program is set to 100H, the declaration of LOADCOM as a procedure is not necessary, and termination is accomplished by simply executing a

```
GO TO 0000H;
```

at the end of the program. Note also that the overhead for a system reboot is not great (approximately 2 seconds), but can be bothersome for system utilities which are used quite often, and do not need the extra space.

The procedures listed in LOADCOM as "library procedures" are a standard set of PL/M subroutines which are useful for CP/M interface. The RELOC procedure contains several nested subroutines for local functions, and actually performs the load operation when called from LOADCOM. Control initially starts on line 327 where the stackpointer is saved and re-initialized to the local stack. The default file control block name is copied to another file control block (SFCB) since two files may be open at the same time. The program then calls SEARCH to see if the HEX file exists; if not, then the high speed reader is used. If the file does exist, it is opened for input (if possible). The filetype COM is moved to the default file control block area, and any existing copies of filename.COM files are removed from the diskette before creating a new file. The MAKE operation creates a new file, and, if successful, RELOC is called to read the HEX file and produce the COM file. At the end of processing by RELOC, the COM file is closed (line 350). Note that the HEX file does not need to be closed since it was opened for input only. The data written to a file is not permanently recorded until the file is successfully closed.

Disk input characters are read through the procedure GETCHAR on line 137. Although the DMA facilities of CP/M could be used here, the GETCHAR procedure instead uses the default buffer at location 80H and moves each buffer into a vector called SBUFF (source buffer) as it is read. On exit, the GETCHAR procedure returns the next input character and updates the source buffer pointer (SBP).

The SETMEM procedure on line 191 performs the opposite function from GETCHAR. The SETMEM procedure maintains a buffer of loaded machine code in pure binary form which acts as a "window" on the loaded code. If there is an attempt by RELOC to write below this window, then the data is ignored. If the data is within the window, then it is placed into MBUFF (memory buffer). If the data is to be placed above this window, then the window is moved up to the point where it would include the data address by writing the memory image successively (by 128 byte buffers), and moving the base address of the window. Using this technique, the programmer can recover from checksum errors on the high-speed reader by stopping the reader, rewinding the tape for some distance, then restarting LOAD (in this case, LOADING is resumed by interrupting with a NOP instruction). Again, the SETMEM procedure uses the default buffer at location 80H to perform the disk output by moving 128 byte segments to 80H through 0FFH before each write.

```

00001 1
00002 1 0FAH: DECLARE BDOS LITERALLY '0005H';
00003 1 /* TRANSIENT COMMAND LOADER PROGRAM
00004 1
00005 1 COPYRIGHT (C) DIGITAL RESEARCH
00006 1 JUNE, 1975
00007 1 */
00008 1
00009 1 LOADCOM: PROCEDURE BYTE;
00010 2 DECLARE FCBA ADDRESS INITIAL(5CH);
00011 2 DECLARE FCB BASED FCBA (33) BYTE;
00012 2
00013 2 DECLARE BUFFA ADDRESS INITIAL(80H), /* I/O BUFFER ADDR
ESS */
00014 2 BUFFER BASED BUFFA (128) BYTE;
00015 2
00016 2 DECLARE SFCB(33) BYTE, /* SOURCE FILE CONTROL BLOCK */
/
00017 2 BSIZE LITERALLY '1024',
00018 2 EOFILE LITERALLY '1AH',
00019 2 SBUFF(BSIZE) BYTE /* SOURCE FILE BUFFER */
00020 2 INITIAL(EOFILE),
00021 2 RFLAG BYTE, /* READER FLAG */
00022 2 SBP ADDRESS; /* SOURCE FILE BUFFER POINTER
*/
00023 2
00024 2 /* LOADCOM LOADS TRANSIENT COMMAND FILES TO THE DISK F
ROM THE
00025 2 CURRENTLY DEFINED READER PERIPHERAL. THE LOADER PLACE
S THE MACH
00026 2 CODE INTO A FILE WHICH APPEARS IN THE LOADCOM COMMAND
*/
00027 2 /* ***** LIBRARY PROCEDURES FOR DISKIO *****
***** */
00028 2
00029 2 MON1: PROCEDURE(F,A);
00030 3 DECLARE F BYTE,
00031 3 A ADDRESS;
00032 3 GO TO BDOS;
00033 3 END MON1;
00034 2
00035 2 MON2: PROCEDURE(F,A) BYTE;
00036 3 DECLARE F BYTE,
00037 3 A ADDRESS;
00038 3 GO TO BDOS;
00039 3 END MON2;
00040 2
00041 2 READRDR: PROCEDURE BYTE;
00042 3 /* READ CURRENT READER DEVICE */
00043 3 RETURN MON2(3,0);
00044 3 END READRDR;
00045 2
00046 2 DECLARE
00047 2 TRUE LITERALLY '1',
00048 2 FALSE LITERALLY '0',
00049 2 FOREVER LITERALLY 'WHILE TRUE',
00050 2 CR LITERALLY '13',

```

```

00051 2      LF LITERALLY '10',
00052 2      WHAT LITERALLY '63';
00053 2
00054 2      PRINTCHAR: PROCEDURE(CHAR);
00055 3          DECLARE CHAR BYTE;
00056 3          CALL MON1(2,CHAR);
00057 3          END PRINTCHAR;
00058 2
00059 2      CRLF: PROCEDURE;
00060 3          CALL PRINTCHAR(CR);
00061 3          CALL PRINTCHAR(LF);
00062 3          END CRLF;
00063 2
00064 2      PRINT: PROCEDURE(A);
00065 3          DECLARE A ADDRESS;
00066 3          /* PRINT THE STRING STARTING AT ADDRESS A UNTIL THE
00067 3          NEXT DOLLAR SIGN IS ENCOUNTERED */
00068 3          CALL CRLF;
00069 3          CALL MON1(9,A);
00070 3          END PRINT;
00071 2
00072 2      DECLARE DCNT BYTE;
00073 2
00074 2      INITIALIZE: PROCEDURE;
00075 3          CALL MON1(13,0);
00076 3          END INITIALIZE;
00077 2
00078 2      SELECT: PROCEDURE(D);
00079 3          DECLARE D BYTE;
00080 3          CALL MON1(14,D);
00081 3          END SELECT;
00082 2
00083 2      OPEN: PROCEDURE(FCB);
00084 3          DECLARE FCB ADDRESS;
00085 3          DCNT = MON2(15,FCB);
00086 3          END OPEN;
00087 2
00088 2      CLOSE: PROCEDURE(FCB);
00089 3          DECLARE FCB ADDRESS;
00090 3          DCNT = MON2(16,FCB);
00091 3          END CLOSE;
00092 2
00093 2      SEARCH: PROCEDURE(FCB);
00094 3          DECLARE FCB ADDRESS;
00095 3          DCNT = MON2(17,FCB);
00096 3          END SEARCH;
00097 2
00098 2      SEARCHN: PROCEDURE;
00099 3          DCNT = MON2(18,0);
00100 3          END SEARCHN;
00101 2
00102 2      DELETE: PROCEDURE(FCB);
00103 3          DECLARE FCB ADDRESS;
00104 3          CALL MON1(19,FCB);
00105 3          END DELETE;
00106 2
00107 2      DISKREAD: PROCEDURE(FCB) BYTE;
00108 3          DECLARE FCB ADDRESS;
00109 3          RETURN MON2(20,FCB);
00110 3          END DISKREAD;

```

```

00111 2
00112 2 DISKWRITE: PROCEDURE(FCB) BYTE;
00113 3 DECLARE FCB ADDRESS;
00114 3 RETURN MON2(21,FCB);
00115 3 END DISKWRITE;
00116 2
00117 2 MAKE: PROCEDURE(FCB);
00118 3 DECLARE FCB ADDRESS;
00119 3 DCNT = MON2(22,FCB);
00120 3 END MAKE;
00121 2
00122 2 RENAME: PROCEDURE(FCB);
00123 3 DECLARE FCB ADDRESS;
00124 3 CALL MON1(23,FCB);
00125 3 END RENAME;
00126 2
00127 2 /* ***** END OF LIBRARY PROCEDURES ***** */
***** */
00128 2
00129 2 MOVE: PROCEDURE(S,D,N);
00130 3 DECLARE (S,D) ADDRESS, N BYTE,
00131 3 A BASED S BYTE, B BASED D BYTE;
00132 3 DO WHILE (N:=N-1) <> 255;
00133 3 B = A; S=S+1; D=D+1;
00134 4 END;
00135 3 END MOVE;
00136 2
00137 2 GETCHAR: PROCEDURE BYTE;
00138 3 /* GET NEXT CHARACTER */
00139 3 DECLARE I BYTE;
00140 3 IF RFLAG THEN RETURN READRDR;
00141 3 IF (SBP := SBP+1) <= LAST(SBUFF) THEN
00142 3 RETURN SBUFF(SBP);
00143 3 /* OTHERWISE READ ANOTHER BUFFER FULL */
00144 3 DO SBP = 0 TO LAST(SBUFF) BY 128;
00145 3 IF (I:=DISKREAD(.SFCB)) = 0 THEN
00146 4 CALL MOVE(80H,.SBUFF(SBP),80H); ELSE
00147 4 DO; IF I<>1 THEN CALL PRINT(.DISK READ ER
ROR$');
00148 5 SBUFF(SBP) = EOF;
00149 5 SBP = LAST(SBUFF);
00150 5 END;
00151 4 END;
00152 3 SBP = 0; RETURN SBUFF;
00153 3 END GETCHAR;
00154 2 DECLARE
00155 2 STACKPOINTER LITERALLY 'STACKPTR';
00156 2
00157 2
00158 2 PRINTNIB: PROCEDURE(N);
00159 3 DECLARE N BYTE;
00160 3 IF N > 9 THEN CALL PRINTCHAR(N+'A'-10); ELSE
00161 3 CALL PRINTCHAR(N+'0');
00162 3 END PRINTNIB;
00163 2
00164 2 PRINTHEX: PROCEDURE(B);
00165 3 DECLARE B BYTE;
00166 3 CALL PRINTNIB(SHR(B,4)); CALL PRINTNIB(B AND 0FH);
00167 3 END PRINTHEX;
00168 2

```

```

00169 2 PRINTADDR: PROCEDURE(A);
00170 3     DECLARE A ADDRESS;
00171 3     CALL PRINTEX(HIGH(A)); CALL PRINTEX(LOW(A));
00172 3     END PRINTADDR;
00173 2
00174 2
00175 2     /* INTEL HEX FORMAT LOADER */
00176 2
00177 2 RELOC: PROCEDURE;
00178 3     DECLARE (RL, CS, RT) BYTE;
00179 3     DECLARE
00180 3         LA ADDRESS,      /* LOAD ADDRESS */
00181 3         TA ADDRESS,      /* TEMP ADDRESS */
00182 3         SA ADDRESS,      /* START ADDRESS */
00183 3         FA ADDRESS,      /* FINAL ADDRESS */
00184 3         NB ADDRESS,      /* NUMBER OF BYTES LOADED */
00185 3         SP ADDRESS,      /* STACK POINTER UPON ENTRY TO REL
OC */
00186 3
00187 3     MBUFF(256) BYTE,
00188 3     P BYTE,
00189 3     L ADDRESS;
00190 3
00191 3     SETMEM: PROCEDURE(B);
00192 4         /* SET MBUFF TO B AT LOCATION LA MOD LENGTH(MBUFF)
*/
00193 4         DECLARE (B,I) BYTE;
00194 4         IF LA < L THEN /* MAY BE A RETRY */ RETURN;
00195 4         DO WHILE LA > L + LAST(MBUFF); /* WRITE A PARA
GRAPH */
00196 4             DO I = 0 TO 127; /* COPY INTO BUFFER */
00197 5                 BUFFER(I) = MBUFF(LOW(L)); L = L + 1;
00198 6                 END;
00199 5             /* WRITE BUFFER ONTO DISK */
00200 5             P = P + 1;
00201 5             IF DISKWRITE(FCBA) <> 0 THEN
00202 5                 DO; CALL PRINT(. 'DISK WRITE ERRORS');
00203 6                 HALT;
00204 6                 /* RETRY AFTER INTERRUPT NOP */
00205 6                 L = L - 128;
00206 6                 END;
00207 5             END;
00208 4             MBUFF(LOW(LA)) = B;
00209 4             END SETMEM;
00210 3
00211 3     READHEX: PROCEDURE BYTE;
00212 4         /* READ ONE HEX CHARACTER FROM THE INPUT */
00213 4         DECLARE H BYTE;
00214 4         IF (H := GETCHAR) - '0' <= 9 THEN RETURN H - '0';
00215 4         IF H - 'A' > 5 THEN GO TO CHARERR;
00216 4         RETURN H - 'A' + 10;
00217 4         END READHEX;
00218 3
00219 3     READBYTE: PROCEDURE BYTE;
00220 4         /* READ TWO HEX DIGITS */
00221 4         RETURN SHL(READHEX,4) OR READHEX;
00222 4         END READBYTE;
00223 3
00224 3     READCS: PROCEDURE BYTE;
00225 4         /* READ BYTE WHILE COMPUTING CHECKSUM */

```

36

```

00226 4      DECLARE B BYTE;
00227 4      CS = CS + (B := READBYTE);
00228 4      RETURN B;
00229 4      END READCS;
00230 3
00231 3      MAKE$DOUBLE: PROCEDURE(H,L) ADDRESS;
00232 4      /* CREATE A BOUBLE BYTE VALUE FROM TWO SINGLE BYTE
S */
00233 4      DECLARE (H,L) BYTE;
00234 4      RETURN SHL(DOUBLE(H),8) OR L;
00235 4      END MAKE$DOUBLE;
00236 3
00237 3      DIAGNOSE: PROCEDURE;
00238 4
00239 4      DECLARE M BASED TA BYTE;
00240 4
00241 4      NEWLINE: PROCEDURE;
00242 5      CALL CRLF; CALL PRINTADDR(TA); CALL PRINTCHAR(':');
;
00243 5      CALL PRINTCHAR(' ');
00244 5      END NEWLINE;
00245 4
00246 4      /* PRINT DIAGNOSTIC INFORMATION AT THE CONSOLE */
00247 4      CALL PRINT('LOAD ADDRESS $'); CALL PRINTADDR(TA);
00248 4      CALL PRINT('ERROR ADDRESS $'); CALL PRINTADDR(LA);
00249 4
00250 4      CALL PRINT('BYTES READ:$'); CALL NEWLINE;
00251 4      DO WHILE TA < LA;
00252 4      IF (LOW(TA) AND 0FH) = 0 THEN CALL NEWLINE;
00253 5      CALL PRINTHEX(MBUFF(TA-L)); TA=TA+1;
00254 5      CALL PRINTCHAR(' ');
00255 5      END;
00256 4      CALL CRLF;
00257 4      HALT;
00258 4      END DIAGNOSE;
00259 3
00260 3
00261 3      /* INITIALIZE */
00262 3      SA, FA, NB = 0;
00263 3      SP = STACKPOINTER;
00264 3      P = 0; /* PARAGRAPH COUNT */
00265 3      TA,LA,L = 100H; /* BASE ADDRESS OF TRANSIENT ROUTINES
*/
00266 3      IF FALSE THEN
00267 3      CHARERR: /* ARRIVE HERE IF NON-HEX DIGIT IS ENCOU
ENTERED */
00268 3      DO; /* RESTORE STACKPOINTER */ STACKPOINTER = SP;
00269 4      CALL PRINT('NON-HEXADECIMAL DIGIT ENCOUNTERED $');
;
00270 4      CALL DIAGNOSE;
00271 4      END;
00272 3
00273 3
00274 3      /* READ RECORDS UNTIL :00XXXX IS ENCOUNTERED */
00275 3
00276 3      DO FOREVER;
00277 3      /* SCAN THE : */
00278 3      DO WHILE GETCHAR <> ':';
00279 4      END;

```

```

00280 4
00281 4      /* SET CHECK SUM TO ZERO, AND SAVE THE RECORD LENG
TH */
00282 4      CS = 0;
00283 4      /* MAY BE THE END OF TAPE */
00284 4      IF (RL := READCS) = 0 THEN
00285 4          GO TO FIN;
00286 4      NB = NB + RL;
00287 4
00288 4      TA, LA = MAKE$DOUBLE(READCS,READCS);
00289 4      IF SA = 0 THEN SA = LA;
00290 4
00291 4
00292 4      /* READ THE RECORD TYPE (NOT CURRENTLY USED) */
00293 4      RT = READCS;
00294 4
00295 4      /* PROCESS EACH BYTE */
00296 4          DO WHILE (RL := RL - 1) <> 255;
00297 4          CALL SETMEM(READCS); LA = LA+1;
00298 5          END;
00299 4      IF LA > FA THEN FA = LA - 1;
00300 4
00301 4      /* NOW READ CHECKSUM AND COMPARE */
00302 4      IF CS + READBYTE <> 0 THEN
00303 4          DO; CALL PRINT(. 'CHECK SUM ERROR S');
00304 5          CALL DIAGNOSE;
00305 5          END;
00306 4      END;
00307 3
00308 3      FIN:
00309 3      /* EMPTY THE BUFFERS */
00310 3      TA = LA;
00311 3          DO WHILE L < TA;
00312 3          CALL SETMEM(0); LA = LA+1;
00313 4          END;
00314 3      /* PRINT FINAL STATISTICS */
00315 3      CALL PRINT(. 'FIRST ADDRESS $'); CALL PRINTADDR(SA);
00316 3      CALL PRINT(. 'LAST ADDRESS $'); CALL PRINTADDR(FA);
00317 3      CALL PRINT(. 'BYTES READ $'); CALL PRINTACDR(NB);
00318 3      CALL PRINT(. 'RECORDS WRITTEN $'); CALL PRINTHEX(P);
00319 3      CALL CRLF;
00320 3
00321 3      END RELOC;
00322 2
00323 2      /* ARRIVE HERE FROM THE SYSTEM MONITOR, READY TO READ THE
HEX TAPE
00324 2
00325 2      /* SET UP STACKPOINTER IN THE LOCAL AREA */
00326 2      DECLARE STACK(16) ADDRESS, SP ADDRESS;
00327 2      SP = STACKPOINTER; STACKPOINTER = .STACK(LENGTH(STACK));
00328 2
00329 2      SBP = LENGTH(SBUFF);
00330 2      /* SET UP THE SOURCE FILE */
00331 2      CALL MOVE(FCBA,.SFCB,33);
00332 2      CALL MOVE(.('HEX',0),.SFCB(9),4);
00333 2      CALL SEARCH(.SFCB);
00334 2      IF (RFLAG := DCNT = 255) THEN
00335 2          CALL PRINT(. 'SOURCE IS READER$'); ELSE
00336 2          DO; CALL PRINT(. 'SOURCE IS DISK$');

```

```

00337 3      CALL OPEN(.SFCB);
00338 3      IF DCNT = 255 THEN CALL PRINT(.'-CANNOT OPEN SOURC
E$');
00339 3          END;
00340 2      CALL CRLF;
00341 2
00342 2      CALL MOVE(.'COM',FCBA+9,3);
00343 2
00344 2      /* REMOVE ANY EXISTING FILE BY THIS NAME */
00345 2      CALL DELETE(FCBA);
00346 2      /* THEN OPEN A NEW FILE */
00347 2      CALL MAKE(FCBA);   FCB(32) = 0; /* CREATE AND SET NEXT RECORD */
00348 2      IF DCNT = 255 THEN CALL PRINT(.'NO MORE DIRECTORY SPACES'
); ELSE
00349 2          DO; CALL RELOC;
00350 3          CALL CLOSE(FCBA);
00351 3          IF DCNT = 255 THEN CALL PRINT(.'CANNOT CLOSE FILES
');
00352 3          END;
00353 2      CALL CRLF;
00354 2
00355 2      /* RESTORE STACKPOINTER FOR RETURN */
00356 2      STACKPOINTER = SP;
00357 2      RETURN 0;
00358 2      END LOADCOM;
00359 1      ;
00360 1      EOF

```


CP/M SYSTEM ALTERATION GUIDE

Table of Contents

Section	Page
1. INTRODUCTION	1
2. FIRST LEVEL SYSTEM REGENERATION	2
3. SECOND LEVEL SYSTEM REGENERATION	6
4. SAMPLE GETSYS AND PUTSYS PROGRAMS	10
5. DISKETTE ORGANIZATION	12
6. THE BIOS ENTRY POINTS	14
7. A SAMPLE BIOS	20
8. A SAMPLE COLD START LOADER	20
9. RESERVED LOCATIONS IN PAGE ZERO	21
10. NOTES FOR USERS OF CP/M VERSION 1.3	23
Appendix	
A. THE MDS LOADER MOVE PROGRAM	
B. THE MDS COLD START LOADER	
C. THE MDS BASIC I/O SYSTEM (BIOS)	
D. A SKELETAL CBIOS	
E. A SKELETAL GETSYS/PUTSYS PROGRAM	
F. A SKELETAL COLD START LOADER	

